SYNTHETIC CIRCUIT GENERATION USING CLUSTERING AND ITERATION

Paul D. Kundarewich and Jonathan Rose

Department of Electrical and Computer Engineering, University of Toronto

Toronto, ON, M5S 3G4, Canada

{kundarew, jayar}@eecg.utoronto.ca

ABSTRACT

The development of next-generation CAD tools and FPGA architectures require benchmark circuits to experiment with new algorithms and architectures. There has always been a shortage of good public benchmarks for these purposes, and even companies that have access to proprietary customer designs could benefit from designs that meet size and other particular specifications. In this paper, we present a new method of generating realistic synthetic benchmark circuits to help alleviate this shortage.

The method significantly improves the quality of previous work by imposing a hierarchy of circuits through clustering and by using a simpler method of characterizing the nature of sequential circuits.

Also, in contrast to current *constructive* generation methods [7-9,11-16,18,19], we employ new iterative techniques in the generation that provide better control over the generated circuit's characteristics. As in previous work, we assess the realism of the generated circuits by comparing properties of real circuits and generated "clones" of the real circuit after placement and routing. On average, the real and clone circuits' total detailed wirelength differ by only 14%, a major improvement over previous results. In addition, the minimum track count is within 14% and the critical path delay is within 10%.

1. INTRODUCTION

There currently exists a shortage of good quality public-domain benchmark circuits that can be used to test the next generation of CAD algorithms for VLSI ASICs and FPGA architectures. Most public domain benchmarks are either too small or not of the right size to give a realistic assessment of the performance of new architectures and algorithms.

The shortage of large circuits exists because companies that possess large circuits regard them as proprietary. A number of efforts have been made to assemble public domain benchmarks, but those that do exist tend to be small or lack crucial information [1-3]. For example, the largest circuits from the MCNC benchmarks [1], common benchmarks used for FPGA research, are on the order of 8000 four-input look-up tables (4-LUT)s. By contrast, the largest planned FPGAs that will be available within a year from Altera and Xilinx have space for up to 114,140 [4] and 111,232 [5] 4-LUTs respectively. This means that the circuits used to evaluate FPGA algorithms and architectures in the research community take up less than 10% of the largest commercial FPGA's area. Furthermore, if the research community is to explore FPGA designs that are five to ten years in the future then circuits that are three to twelve times larger are needed as chip size is forecast to grow by that amount according to the International Technology Roadmap for Semiconductors [6]. When faced by such a disparity between the size of the next generation of FPGA designs and the size of circuits used to explore these designs one has to wonder how realistic are conclusions reached with such circuits.

This shortage of larger circuits is even more acute when one considers that what is really needed to fully test FPGA architectures or algorithms are larger circuits of the right size. In testing, a circuit that consumes half the logic resources of the target FPGA is often not as interesting as a circuit that consumes 90% of an FPGA and places large demands on the architecture or the CAD tool algorithms. Furthermore, unlike research into ASICs, if the benchmark circuit does not fit into the FPGA, then that benchmark circuit is of no use for testing purposes.

Recently, researchers have proposed several approaches to *synthetic* benchmark generation in an attempt to alleviate these problems. Synthetic benchmarks are netlists created by an automated program and are constrained to have a specific set of desirable characteristics. However, for synthetic circuits to be useful, they must be shown to be realistic proxies for real circuits.

Hutton et al. [7,8] demonstrated realism by comparing real benchmark circuits to "clones" generated synthetically from the characterization of the real circuits. Real and clone circuits could be compared on the basis of an important circuit characteristic or property (such as power consumption, critical path delay or total wirelength after placement and routing). They were successful in generating good quality clones (as measured by wirelength) of circuits that were purely combinational, but the approach worked less well for the larger sequential circuits, producing circuits that require 40% more wirelength on average.

In this paper, we propose several new characterization parameters and synthetic generation techniques that significantly improve upon the wirelength results of Hutton et al. [7,8] for sequential circuits. At the same time, we maintain the key strength of that work—the ability to directly specify the unit delay profile of the synthesized circuits. Finally, we view our work as a key (but not final) step towards the goal of having the ability to create larger circuits than already exist.

This paper is organized as follows: in Section 2 we review prior literature in synthetic circuit generation, including Hutton et al. [7], upon which this research is based. Section 3 describes a set of new characterization parameters that we propose to improve the generation results. Section 4 describes the new generation methods. In Section 5, we present measurements of the quality of the generated circuits, and conclude in Section 6.

2. BACKGROUND AND PREVIOUS WORK

This chapter provides a review of other circuit generators that exist in the literature. This is followed by the background of the work of Hutton et al. [7,8] upon which this research is based.

2.1 Other Synthetic Generation Efforts

A number of other synthetic circuit generators have recently been proposed in the literature [9,11-19], which we will now review.

Darnauer and Dai [9] generate synthetic circuits with a fixed number of inputs, outputs, LUTs, and with an average fanin and approximate Rent exponent [10]. The method constructs the synthetic circuit by recursively bi-partitioning the circuit and making connections after each partition until the clusters consist of single LUTs. The method is notable in its attempt to capture the hierarchical nature of a circuit. However, the approach lacks control over the fanout and unit delay profile of gates in the circuit. No validation is done on the quality of the synthetic circuits generated as the work focused on determining the routability of the synthetic circuits with given input parameters.

Iwama et al. [11] create synthetic benchmark circuits using functional transformations of pre-existing real circuits that preserve the logical function of the circuit. While the resulting circuit is “realistic” from a logical standpoint, the method suffers from a lack of control over the physical properties of the netlist and the need for prior circuits. In [12] the method was extended to limit the fanin values of gates in the circuits but the lack of control over physical properties still remains. To alleviate the need for a prior circuit, a trivial random circuit generator was also described that generates single sum-of-product term circuits. No validation was done on any of the synthetic circuits as the application of their work was towards evaluating the ability of synthesis tools to reoptimize circuits they transform.

Harlow and Brglez [13] and Ghosh et al. [14,15] propose the idea that synthetic circuits should be generated by characterizing properties about circuits that will remain invariant under transformations or "mutations" of the circuits. The goal of their work is to provide synthetic circuits to help the influence that different starting conditions have on the testing of CAD algorithm performance. They claim that by expanding each circuit used in testing into an equivalence class of mutant circuits and averaging over the experimental results they can negate the effect of different starting conditions. The advantage of this approach is that the circuits are not completely random. However, the approach does not lend itself to the possibility of scaling the mutants to larger circuit sizes, which is a key motivation behind synthetic circuit generation. Furthermore, they present their results based on a small number of small-sized circuits and thus it is unclear how their method performs for larger circuits.

Pistorius et al. [16] characterize digital designs as consisting of two levels of hierarchy with five different types of logic. At the bottom level of the hierarchy are regular combinational logic, irregular combinational logic, memory blocks, and combinational and sequential logic. At the top level of the hierarchy is the interconnection logic connecting these different sub-circuits. Generators are proposed for the regular combinational logic, the memory, the combinational and sequential logic, and the interconnection logic. Success is judged in the context of partitioning multiple FPGA systems. Here the utilization of the FPGAs is a key concern, and their method achieves an average filling rate for the clones that deviated by less than 17% from the original circuits. Success is not judged on the basis of the wirelength or delay properties and it is unclear as to whether under these latter criteria the circuits would prove realistic. No characterization or method to judge success is given at the second level of hierarchy.

Wilton et al. [17] generate synthetic circuits with both logic and memory. Their method of generation is, first, to characterize large numbers of real circuits with a view to how logic and memory interconnect and the number, size, and shapes of the memories. Second, synthetic circuits are generated stochastically by randomly selecting a memory configuration and interconnect pattern from the characterization, randomly selecting combinational logic from the MCNC benchmarks [1] and making connections between the memory and logic based on the interconnect pattern. The strength of the method is the inclusion of memory in the final circuits and the realistic connections between memory and logic. The weakness of the method is its use of only combinational MCNC benchmarks in logic portions of the generated circuits.

Stroobandt et al. [18] developed a synthetic benchmark generation method that generates circuits using a bottom up clustering approach. This method produces circuits that are too regular and that have unrealistic delay profiles [19]. Verplaetse et al. [19] attempt to fix these problems and achieves good wirelength results with the real circuit and clones differing by 7% on average. It is unclear whether or not the delay profile problem has been sufficiently fixed since no direct comparison is made between the synthetic circuit delay profile and that of a real circuit.

Hutton et al. [7,8] can generate synthetic circuits that scale with size and their generation method provides direct control over the unit delay profile of the synthesized circuits. Of all the synthetic benchmark efforts we feel that it shows the most promise and it is what we base our work upon. We now review their work.

2.2 Hutton et al.'s Characterization and Generation

The synthetic circuit generation approach of Hutton et al. [7] is to characterize key physical circuit properties of combinational circuits and then to generate synthetic circuits that are constrained to have these properties. When the properties of the original circuits are unchanged, these generated circuits are called *clones* of the original circuit. Examples of physical properties include the fanout distribution of the gates, the delay structure, and the number and type of connections in the circuit. In [8], Hutton et al. extended their method to generate sequential circuits, with flip-flops. In the sections below, we first define the combinational circuit model and definitions. Second, we discuss the set of circuit characterizations and the method of combinational synthetic circuit generation. Third, we provide a brief description of the extensions made for sequential circuits. Lastly, we describe the process by which the circuits are judged realistic and reprise Hutton et al.'s results.

2.2.1 Circuit Models and Definitions

Circuits are modeled as a directed acyclic graph $G = (V,E)$ where the nodes V represent gates in the circuit and edges E represent two-point connections between gates. In order to reduce the wide variation of gate types, both Hutton et al. [7] and this work assume that all gates are 4-input lookup tables (4-LUTs). As a key aspect of circuits is their delay, Hutton et al. employs the unit delay model in which every LUT incurs a single unit of delay.

With this delay model, the *delay level* of a node in the graph is defined as the maximum delay over all directed paths beginning at a primary input (PI) or a flip-flop (DFF) and terminating at the given node. The maximum combinational delay over all nodes in a circuit is defined as d_{max}.

The delay structure of the circuit is characterized by a collection of measurements at the various delay levels. *Shape* is defined as the number of objects at each delay level. Accordingly, Hutton et al. [7] defines *node shape*, *input shape*, *output shape*, and *PO shape* as the total number of nodes, inputs, outputs, and primary outputs (POs) at each delay level respectively. The concept of shape is illustrated in Figure 1 for the MCNC circuit cm151a. The illustration of the circuit shows the nodes arrayed and labeled by delay level. Figure 1 also gives a histogram of the node, input, and output shapes. Primary inputs, which occupy the 0^{th} delay level, are labelled PI. Looking at the 1st delay level we see it has 4 nodes, 16 inputs from the 0^{th} delay level, and 4 outputs, and no primary outputs.

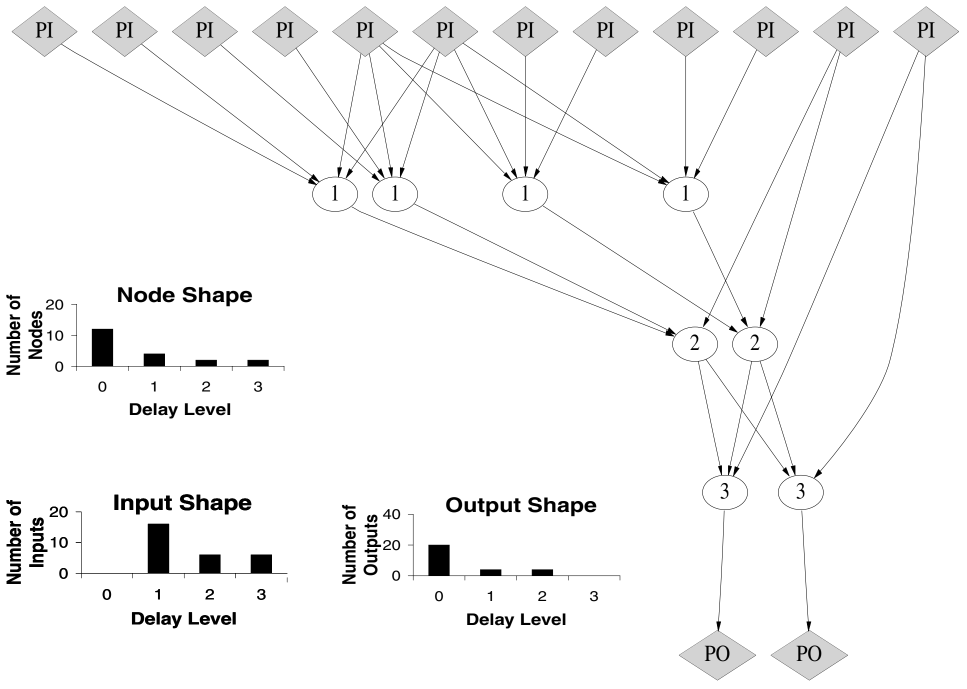

Figure 1 - Circuit and Shape Functions

For a node *x*, *fanout(x)* is the combinational output degree of the node. For a circuit, Hutton et al. [7] describes the fanout in terms of the *fanout distribution,* defined as the number of nodes of each fanout, starting at 0.

To characterize the connections in the combinational circuit, Hutton et al. [7] defines an *edge length* property: For an edge *e=(x,y)* with nodes *x* and *y* they define the *length(e) = delay_level(y) - delay_level(x)* if *delay_level(y) > delay_level(x)*. An edge of length 1 is termed a *unit edge* while any edge with a length greater than 1 is termed a *long edge.* Using this definition of length they define the *edge length distribution* as the number of edges at each edge length.

2.2.2 Combinational Generation Algorithms

The method Hutton et al. [7] used to generate combinational circuits proceeds in a few basic steps as illustrated in Figure 2. The input into the generation phase is the node shape, the fanout distribution, the edge length distribution and several other parameters that are omitted here for simplicity (i.e. the number of primary inputs, the number of primary outputs, the maximum fanin to a LUT, and the locality parameter “L”). In the algorithm, nodes are organized by delay level into larger groupings called *level nodes.* In Step I, the input shape's and output shape's upper and lower bounds are computed at each level node. In Step II, the majority of edges are assigned between the level nodes. In Step III, the fanout distribution is partitioned among the level nodes. In Step IV, each level node is split into individual nodes with a specific fanout. In Step V, edges are assigned between individual nodes.

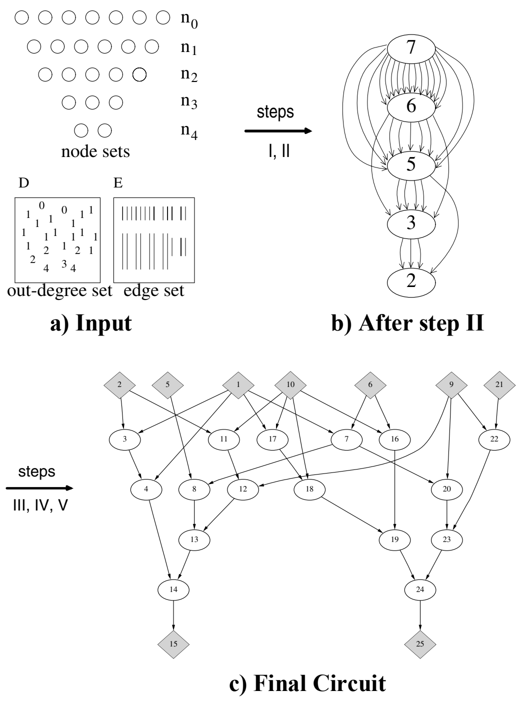

Figure 2 - Combinational Generation Algorithm [7]

It is at this point that Hutton et al. [7] attempts to achieve realistic wirelengths in the final generated circuit by imposing a notion of "locality" on the edge assignments. Here each node at each delay level is assigned a horizontal position, and edges are chosen in such a way as to minimize the horizontal distance between joined nodes.

The output from the process, ideally, is a graph where the specified size, distributions and shape functions are met. For example, each node should have a fanout value from the fanout distribution that matches its number of output edges and each node should belong to the correct delay level as dictated by the shape function. Furthermore, there should be no node “violations”, which are nodes that have one or more of the following properties: no outputs, no inputs, too many inputs, or two or more connections from the same source node. Hutton et al. [7] showed that it was often difficult to precisely meet the specification as given, and that it was difficult to prevent all node violations. In order to deal with this, they reduced the length of edges and fanout values, dropped the edges during Step V that could not find valid connections, and assigned primary outputs to nodes with no output edges.

It is relevant to note that this generation approach was largely constructive: at each Step, an assignment is made based on a calculated ordering, which was in turn based on a specific cost metric. In the present work, we propose a new method that iterates over various states of the fully constructed graph employing an ensemble cost function that measures the degree of success in meeting the generation specifications.

2.2.3 Extensions for Sequential Circuits

In [8], Hutton et al. extended the above method of generating combinational circuits to sequential circuits. Here sequential circuits were broken into a set of combinational sub-circuits separated by flip-flops that could be characterized and generated separately and then "glued" together during generation to form a full circuit.

The combinational sub-circuits are identified by thinking of sequential circuits as consisting of chains of combinational logic that are connected to the next stage by flip-flops and are connected to any previous stage by feedback connections. To find these combinational sub-circuits the nodes in a circuit are partitioned into groups, termed *sequential levels,* based on the *sequential level numbers* of the nodes. The sequential level number of a node *x*, *sequential_level (x)*, is defined as *0* if *x* is a Primary Input, *1 +*

sequential_level (y) for a flip-flop *x* with input *y*, and *MIN(sequential_level(y_i))* over all inputs y_i to *x* otherwise. In combinational circuits, there is a single sequential level. In sequential circuits, the sequential levels form a hierarchy. An example of this abstract model of a sequential circuit is given in Figure 3 [8].

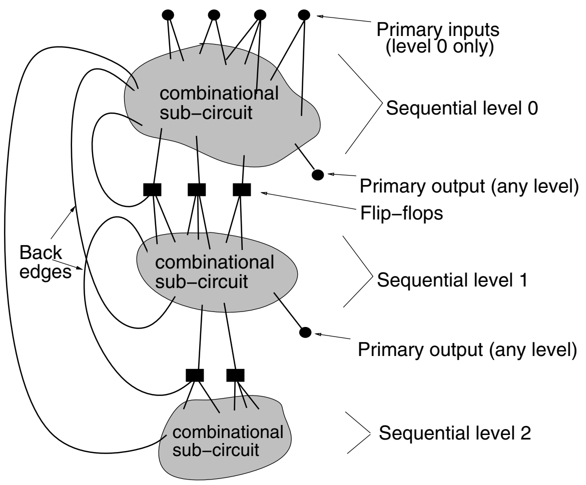
Primary inputs (level 0 only)
combinational sub–circuit
Sequential level 0
Primary output (any level)
Flip–flops
Back edges
combinational sub–circuit
Sequential level 1
Primary output (any level)
combinational sub–circuit
Sequential level 2

Figure 3 - Hutton et al.'s Model of a Sequential Circuit [8]

With the circuit broken into a hierarchy of sequential levels, Hutton et al.'s [7] combinational characterization is applied to each sequential level with extra characterization added to model flip-flop and feedback connections that cross sequential level boundaries. Hutton et al. [8] characterizes the connections that enter or leave each sequential level with the *ghost input shape* (GIshape) defined as the number of edges entering a sequential level at each delay level and the *ghost output shape* (GOshape) defined as the number of edges that exit from a sequential level at each delay level.

Hutton et al. [8] generates synthetic sequential circuits in two steps: In Step I, the combinational logic at each sequential level is generated separately by the method described in Section 2.2.2 above with modifications made to choose the set of nodes for flip-flop connections and the set of nodes for feedback connections from the ghost input and output shapes. In Step II, the sequential levels are connected together by connecting each sequential level to the flip-flops at the next sequential level and then by connecting each sequential level to the previous sequential levels by randomly making feedback connections. In Step II, the nodes used in flip-flop connections or feedback connections are chosen from the respective sets.

The key disadvantage of Hutton et al.'s [8] sequential circuit model is many circuits don't have the pipeline-only structure implied by the sequential levels in this model. Certainly, the pipeline portion of circuits have this structure, but all others do not.

2.2.4 Quality of Previous Circuits

In Hutton et al. [7,8] the quality of the generated circuits was judged by comparing real circuits and clone circuits generated from the characterizations measured from each real circuit. This process of judging real circuits against their clones is called *validation* and is illustrated in Figure 4. It is the framework we will use to judge the quality of our synthetic circuits. Note, because we directly compare the circuit characteristics of the clones against that of the real circuits this approach is a direct validation approach versus an indirect validation approach as defined in [20].

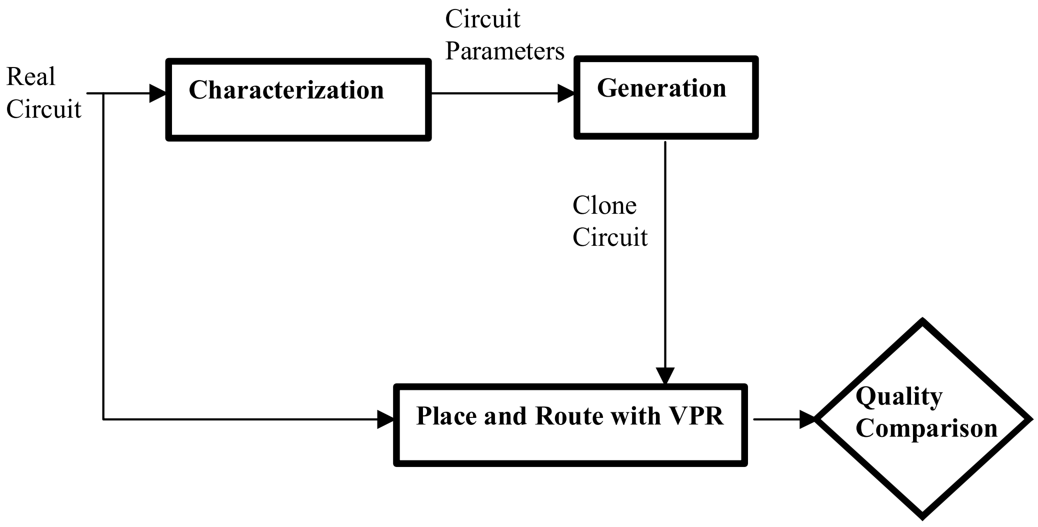

Figure 4 - Validation Process

In Hutton et al. [7,8] the real and clone circuits were compared on the basis of wirelength achieved after placement and global routing. Circuits were placed and global routed using VPR [21,22] and the wirelength measured. It was found that post-placement wirelength differed by 17% on average for purely combinational circuits and 40% on average for sequential circuits (which were generally larger than the combinational circuits). Note that Hutton et al. measured the average of the absolute value of the difference between the original and clone circuits. Also, for the sequential circuits, in almost all cases the wirelength of the clone circuit was *greater* than that of the original circuit.

While the quality of combinational synthetic circuits was reasonably good, the quality of sequential circuits is too different from the original circuits to be used as reasonable proxies in FPGA architecture development. Hutton et al. [8] suspects that the reason behind the wirelength differences is that the generated circuits lack a hierarchy that can be observed in typical circuits. In this paper, we directly address this issue.

3. NEW CIRCUIT MODEL AND CHARACTERIZATION

In order to introduce hierarchy into synthetic circuits we identify that hierarchy through clustering. To characterize the result of a clustering, we define a new model that describes clusters and various aspects of their connectivity. As discussed above, this new model will be employed in the context of Hutton et al. [7,8] with new circuit characterizations and new generation techniques.

3.1 New Circuit Model for Sequential Circuits

As described in Section 2.2.3, Hutton et al.'s [8] sequential circuit model is not very natural for all circuits. Here we choose a simpler circuit model that makes it easier for a hierarchy to be identified. It is similar to Hutton et al.'s [7] combinational circuit model described in Section 2.2.1 but with three additions made to account for flip-flops. The first addition is that flip-flops are placed at delay level 0, the same as the primary inputs. The outputs of these flip-flops drive into the combinational logic just as primary inputs do. Second, the flip-flops themselves must be driven. To do so, some of the regular combinational nodes are designated as *latched nodes*, meaning that the node's output drives the data input of a flip-flop. Finally, this edge that joins a latched node to its flip-flop ("a DFF edge") is defined to have an edge length of 0. An example of this new circuit model is depicted in Figure 5.

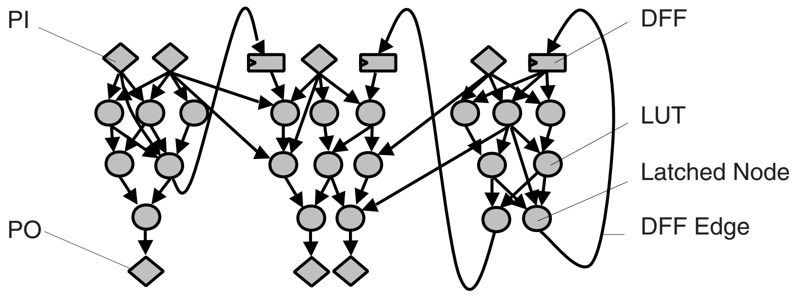

Figure 5 - New Circuit Model

These changes to the circuit model remove the pipeline-like assumption of the previous model, but still permit a well-defined characterization of all sequential circuits with a single clock. The new circuit model is also simpler, making it easier to identify a hierarchy (described in the next section) and to perform iterative-based generation (described in Section 4).

3.2 Identifying a Hierarchy

To identify a hierarchy in a circuit (while characterizing a pre-existing circuit) we partition the circuit into clusters. Figure 6 shows the sample circuit of Figure 5 partitioned into three connected clusters. The clusters in the circuit are defined given the graph of the circuit $G = (V,E)$ as resulting from a partition of V into a series of k clusters $C_1, C_2, \ldots C_k$ where the k-clusters are disjoint vertex sets that fully cover V.

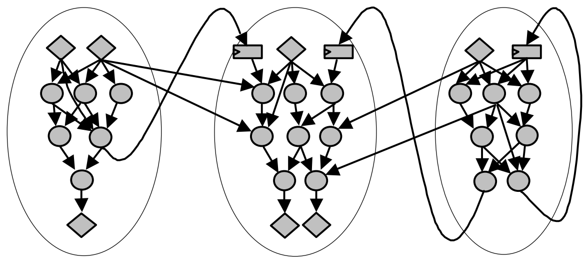

Figure 6 - A Circuit in Three Clusters

3.3 Characterization of Clustered Circuits

We break the description of the new characterization into four sections. First, we describe the characterizations that we apply unchanged from Hutton et al.'s [7,8] work. Second, we describe the characterization of the connections between clusters. Third, we describe the new characterizations needed to deal with changes made necessary by the fact that the circuits are no longer single cluster systems. Lastly, we describe a characterization of an approximation to wirelength that we will use to control wirelength in the synthetic circuits that are generated. Throughout our description of our characterization we will use the circuit in Figure 6 as an example.

3.3.1 Unchanged Characterization

We keep unmodified from Hutton et al.'s [7,8] characterizations the number of nodes, the number of primary inputs, the number of flip-flops, the node shape, Primary Output (PO) shape, and fanout distribution. These characterizations are applied to each cluster separately. These values for the circuit in Figure 6 are summarized in Table 1.

Table 1 - Unchanged Characterization for Figure 6

Characterization	Cluster 1	Cluster 2	Cluster 3
Number of Nodes	8	11	9
Number of PI	2	1	1
Number of Flip-Flops	0	2	1
Node Shape	(2 3 2 1)	(3 3 3 2)	(2 3 2 2)
PO Shape	(0 0 0 1)	(0 0 0 2)	(0 0 0 0)
Fanout Distribution	(1 3 2 1 0 1)	(2 2 2 1 2 0)	(2 2 2 1 2 0)

3.3.2 Inter-cluster Characterization

The structure of the connections between each pair of clusters $C_1..C_k$ is captured through two matrices that count the number of connections to combinational nodes and flip-flops between clusters. The first matrix we define as *Comb=[comb$_{ij}$]* where *comb$_{ij}$* is the number of inter-cluster connections that drive combinational nodes from clusters *C_i to C_j*. The second matrix we define as *Latched=[latch$_{ij}$]* where *latch$_{ij}$* is the number of connections that drive flip-flops from *C_i to C_j*. Figure 7 and 8 illustrate the inter-cluster connections that each of these matrices separately capture for the circuit in Figure 6. We have separated the inter-cluster connectivity into these two matrices because we found that they are weakly correlated [23] and because it allows the generation of the combinational and sequential structure separately. The values of the *Comb* and *Latch* matrices for the circuit in Figure 6 can be seen in Figure 9 and Figure 10 respectively.

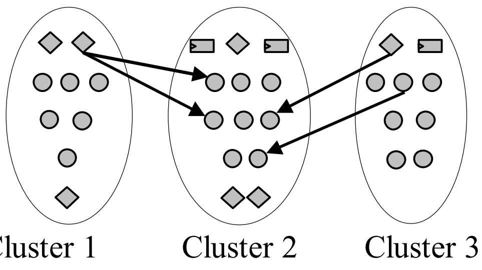

Figure 7 - Inter-cluster Connections to Combinational Nodes

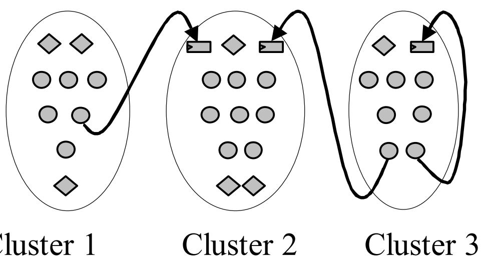

Figure 8 - Inter-cluster Connection to Flip-flops

$$\begin{array}{cc} & \text{Sink Cluster} \\ \text{Source Cluster} & \begin{array}{c}\begin{array}{cccc} & 1 & 2 & 3 \end{array}\\ \begin{array}{c} 1 \\ 2 \\ 3 \end{array}\begin{bmatrix} 0 & 2 & 0 \\ 0 & 0 & 0 \\ 0 & 3 & 0 \end{bmatrix}\end{array}\end{array}$$

Figure 9 – *Comb* for the Circuit in Figure 6

$$\begin{array}{cc} & \text{Sink Cluster} \\ \text{Source Cluster} & \begin{array}{c}\begin{array}{cccc} & 1 & 2 & 3 \end{array}\\ \begin{array}{c} 1 \\ 2 \\ 3 \end{array}\begin{bmatrix} 0 & 2 & 0 \\ 0 & 0 & 0 \\ 0 & 3 & 0 \end{bmatrix}\end{array}\end{array}$$

Figure 10 – *Latch* for the Circuit in Figure 6

3.3.3 New Intra-cluster Characterization

Inside each cluster, we add additional characterizations for the input and output shapes, the latched shape, and the edge length distribution.

The first addition to the circuit characterization was to explicitly include input and output shapes– the number of inputs and outputs entering each combinational delay level. Hutton et al.'s [7,8] generation method derived bounds for these values, but we instead include it explicitly as his process was too complex with the inclusion of multiple clusters.

The second addition is the *latched shape*, which is defined as the number of nodes connected to flip-flops at each delay level. For example, for Cluster 1 of Figure 6 we can see that only one node from the 2nd delay level is connected to a flip-flop and so it has a latched shape of (0 0 1 0).

The last addition was to modify the edge length distribution to account for inter-cluster edges not present in Hutton et al.'s [7,8] work. The inter-cluster edges need to be characterized because they range from 5 to 45% of the total number of edges in the circuit and thus have a large impact on the circuit structure. We characterize the edge length distribution in three parts: the *intra-cluster edge length distribution, the inter-cluster input edge length distribution,* and *the inter-cluster output edge length distribution* defined as the number of edges at each edge length internal to the cluster, that input into the cluster, and that output out of the cluster respectively. Figure 11 illustrates the concept of intra-cluster, inter-cluster input, and inter-cluster output edges.

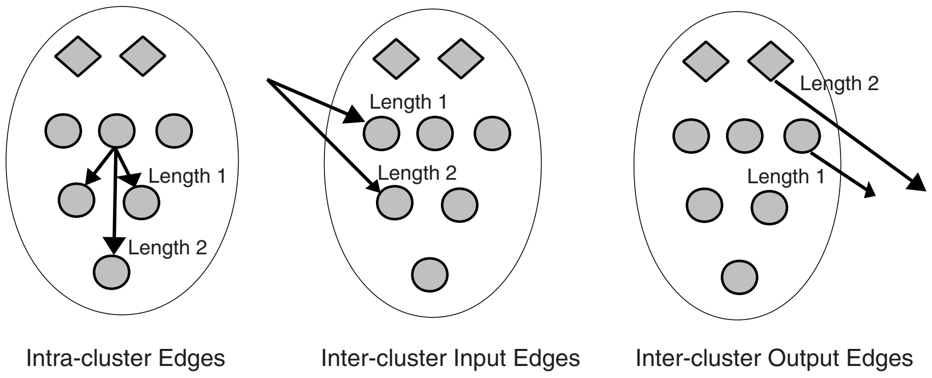

Figure 11- Intra- and Inter- Cluster Edges

The input shape, output shape and edge length distributions for the circuit in Figure 6 are summarized below.

Table 2 – New Characterizations for the Circuit in Figure 6

Characterization	Cluster 1	Cluster 2	Cluster 3
Input Shape	(0 5 6 2)	(0 6 7 5)	(0 6 4 5)
Output Shape	(8 5 2 0)	(6 4 4 0)	(7 6 4 0)
Latch Shape	(0 0 1 0)	(0 0 0 0)	(0 0 0 2)
Intra-cluster edge length dist.	(1 12 1 0)	(2 13 1 0)	(2 14 1 0)
Inter-cluster input edge length dist.	(0 0 0 0)	(0 1 3 0)	(0 0 0 0)
Inter-cluster output edge length dist.	(0 1 1 0)	(0 0 0 0)	(0 0 2 0)

3.3.4 Wirelength Characterization

To control the post-place and route wirelength of the synthetic circuits that we produce from our generation process (described in Section 4) we measure an approximation to wirelength in characterization that we will use in generation. The approximation is a metric that Hutton defined but never used in generation [24]. First, we will describe the metric and its motivation. Second, we will describe an algorithm to measure it.

Hutton [24] used an approximation of wirelength instead of real post-place and route wirelength because he wanted to quickly characterize a small amount of information about local structure and thought that a full placement and routing of a circuit on an FPGA or ASIC would be too computationally expensive. Instead, he "placed" the graph within the combinational delay graph structure by assigning each node at each delay level a horizontal position and ordering the horizontal position so as to minimize the number of edges that crossed and the horizontal distance between connected nodes. In our modification of this placement algorithm, we minimize only the total horizontal distance.

With the circuit "placed", Hutton [24] measured his approximation to wirelength that he defined as:

$$wirelength_{Approx}(G) = MIN\Big(\sum_{x \in V(G)} \sum_{y \in Inputs\ of\ X} \left| horizontal\_position(y) - horizontal\_position(x) \right| \qquad (1)$$

With this approximation we will control the wirelength of the synthetic circuits we produce in generation.

3.4 Software Implementation of Characterization

We rewrote Circ the software tool that Hutton [7,8] built to characterize circuits. The new tool, called CCirc, takes as input a circuit in the BLIF [1] netlist format and outputs statistical information corresponding to all of the characterizations discussed above, into a "stats" file.

CCirc uses a partitioner to identify a hierarchy in a circuit. We employed the hMetis partitioning package [25] to divide the circuit into clusters because it is a well-regarded partitioner that is freely available. It also possesses an easy-to-use API that can be called from within CCirc. hMetis is a multilevel min-cut partitioner and can partition circuits using a recursive bi-partition or k-way method with different node balancing conditions.

The source code and executables for CCirc can be found at:

http://www.eecg.toronto.edu/~jayar/software/Cgen/Cgen.html

4. GENERATION

We now describe a method of generating synthetic circuits. Its three key features are that i) it employs the new sequential model described in Section 3.1, ii) it generates circuits as groups of connected clusters as described in Section 3.2, and iii) its overall approach is to use *iteration* in the generation process, as opposed to the constructive approach taken by Hutton et al. [7,8].

The input to the generation process is the characterizations we defined in the previous section which are: the *Comb* and *Latched* matrices, the number of clusters, and for each cluster the number of nodes, the number of primary inputs, the number of flip-flops, the node shape, the primary output shape, the latched shape, the input shape, the output shape, the fanout distribution, the intra-cluster edge length distribution, and the inter-cluster input and output edge length distributions. The output from the generation process is a circuit in BLIF [1] or structural VHDL format ready to be placed and routed. We set the function of each LUT to be a NAND gate to give each LUT a logic type.

The generation algorithm proceeds in four steps. In Step 1, we create the delay structure of the circuit by assigning edges between the level nodes in the circuit (recall that a level node contains all of the individual nodes at each delay level). In Step 2, the fanout distribution in each cluster is partitioned among the level nodes. In Step 3, the level nodes are split into individual nodes and the individual nodes are prepared for final edge assignment. In Step 4, edges are assigned between individual nodes in the circuit based on the delay structure of the circuit and the fanouts assigned to the individual nodes. We have broken the generation process into these four steps because we felt that trying to satisfy all the requirements all at once would be too computationally difficult and complex. At each stage of the algorithm, as an example, we will follow the generation of a synthetic circuit using the characterization given in Section 3 of the circuit given in Figure 6

4.1 Creation of Delay Structure

In the first step, we create the delay structure of a circuit. Individual nodes in the delay levels of a cluster are aggregated into level nodes. Edges between the level nodes are aggregated into *super edges* where a super edge is an edge between two levels nodes with a weight equal to the number of individual edges between the two level nodes. The grouping of the nodes and edges into larger aggregates allows us to first concentrate on the large-scale connectivity between delay levels in the clusters of the circuit before trying to satisfy other requirements such as the fanout distribution or individual edge assignment.

The input to this step is the *delay structure characterization* which consists of the *Comb* and *Latched* matrices, d_{max}, and for each cluster the node shape, the input shape, the output shape, the latched shape, the intra-cluster edge length distribution, the inter-cluster input edge length distribution, and the inter-cluster output edge length distribution.

The desired output is the delay structure with the weights of the super edges assigned to ensure that each individual node will be able to have its delay level correctly set, the delay structure will not force node violations to be made during final edge assignment, and deviations from the given characterization are minimized

Figure 12 illustrates the basic input into and output from delay structure creation. Figure 12a shows the individual nodes aggregated into level nodes. The desired numbers of input and output edges for each level node have been annotated from the Input and Output Shapes. Figure 12b shows target edge length distributions that our algorithm will attempt to satisfy. Figure 12c shows the inter-cluster matrices *Comb* and *Latch* in graph form that our algorithm will attempt to satisfy. The output of the algorithm is shown in Figure 12d. Here, the number of edges between the various level nodes (indicated by the edge weight) has been determined.

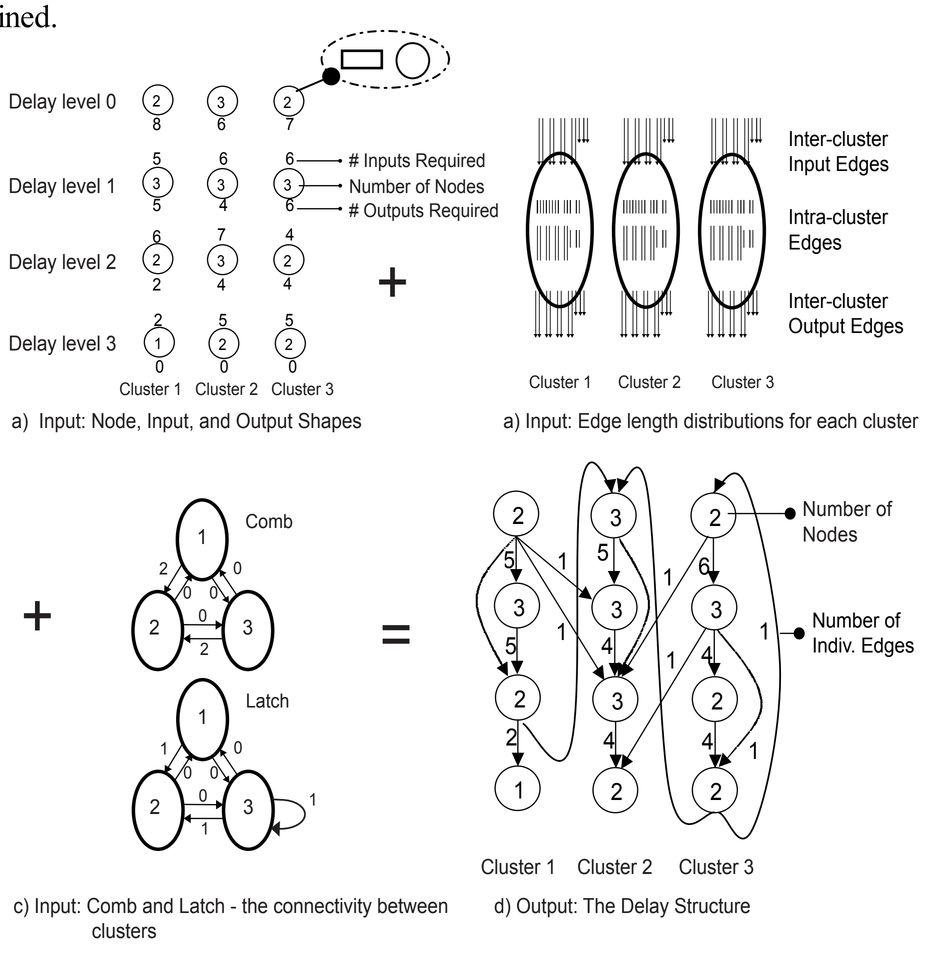
Delay level 0
Delay level 1
Delay level 2
Delay level 3
# Inputs Required
Number of Nodes
# Outputs Required
Cluster 1
Cluster 2
Cluster 3
a) Input: Node, Input, and Output Shapes
Inter-cluster Input Edges
Intra-cluster Edges
Inter-cluster Output Edges
a) Input: Edge length distributions for each cluster
Comb
Latch
c) Input: Comb and Latch - the connectivity between clusters
Number of Nodes
Number of Indiv. Edges
d) Output: The Delay Structure

Figure 12 – Creation of the Delay Structure

The algorithm for solving this problem is divided into two parts. The first part creates the combinational connections in the circuit while the second part creates the sequential connections in the circuit making connections from level nodes with latched nodes to level nodes with flip-flops. We describe each of these parts separately.

4.1.1 Creating the Combinational Delay Structure

We create the initial solution to the Combinational Delay Structure by inserting all intra- and inter- cluster edges into the graph. Input into the algorithm is the delay structure characterization without the latched shapes or *Latched* matrix. Output from the algorithm is the Combinational Delay Graph Structure, an example of which is shown in Figure 13. The figure shows the level nodes in each cluster (and the number of individual nodes contained in a level node), the weight assigned to the super edges.

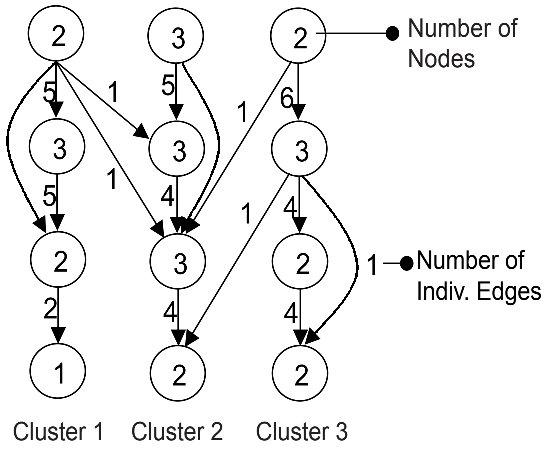

Figure 13 – Combinational Delay Structure

The edges are inserted into the graph based on trying to satisfy the edge length distributions, *Comb*, the input shape, the output shape, and on making sure that each level node has enough unit edges to define the delay level of its nodes. The delay level of a node is defined if it has a unit edge from the delay level above. More details about this and all phases of this algorithm can be found in [23].

After creating the initial solution, we employ an iterative algorithm that selects certain edges as candidates for relocation and accepts or rejects proposed changes based on a cost function.

In the following paragraphs, we will describe the cost function, how edges are selected and modified ("moves"), the overall structure of the algorithm, and the performance of the algorithm.

The cost function of the iterative algorithm is as follows:

$$cost_{Delay\ structure} = \mathrm{cost}_{Comb} + cost_{Edge\ Length} + cost_{Level\ Shape} + cost_{Problem\ Node} \tag{2}$$

Here, $cost_{Comb}$ measures the difference between the current *Comb* and its specification by subtracting the two matrices and adding the absolute values of the entries of the matrix, $cost_{EdgeLength}$ measures the absolute difference between the current edge length distributions and their specifications, $cost_{LevelShape}$ measures the absolute difference between the current input and output shapes and their specifications with congestion factors multiplying this cost at each level node to penalize level nodes that have too many inputs or outputs. Finally, $cost_{Problem\ Node}$ measures the number of node violations that would be forced to be made during final edge assignment because of the number of edges that input into or output out of a level node.

We selects super edges to change in the graph as in Figure 14.

```
Randomly select a source edge source ∈ E(G)

Randomly select destination edge dest ∈ E(G) s.t.

     length(source) = length(dest) && weight(dest) < max_weight(dest)
```

Figure 14 – Move Generation

We employ an iterative improvement algorithm, in which all changes that improve the cost function are accepted, and bad moves are accepted with a probability of $e^{-\Delta \cos t/6}$. The algorithm continues until the cost is zero or until the number of moves attempted is fifty times the number of edges in the graph.

We can give an idea of the success of the optimization. The cost function measures the deviance of the result from the specification. We normalize the cost by dividing it by the number of edges in the graph. The optimizer is typically able to reduce $cost_{Delay\ Structure}$ to about 5% of the total number of edges. Of this 5%, $cost_{Comb}$ usually makes up 2%, $cost_{EdgeLength}$ usually makes up 26%, $cost_{Level\ Shape}$ usually makes up 67%, and $cost_{Problem\ Node}$ usually makes up 4%.

At the end of iteration, it is possible that node violations (which are nodes that have one or more of the following properties: no outputs, no inputs, too many inputs, or nodes with two or more connections from the same source node) remain. In this case we post-process the graph by removing or adding edges to remove the violations.

4.1.2 Creating the Sequential Delay Structure

With the global combinational delay structure complete, the delay structure is finished by forming the connections between the level nodes with latched nodes and the level nodes with flip-flops. The input into this phase of the algorithm is the matrix *Latched* and the latched shapes in each cluster. The output is the finished delay structure graph. A picture of the completed delay graph structure can be seen in Figure 15, which shows the addition of the latched node to flip-flop connections. Our algorithm to create these connections is given in Figure 16.

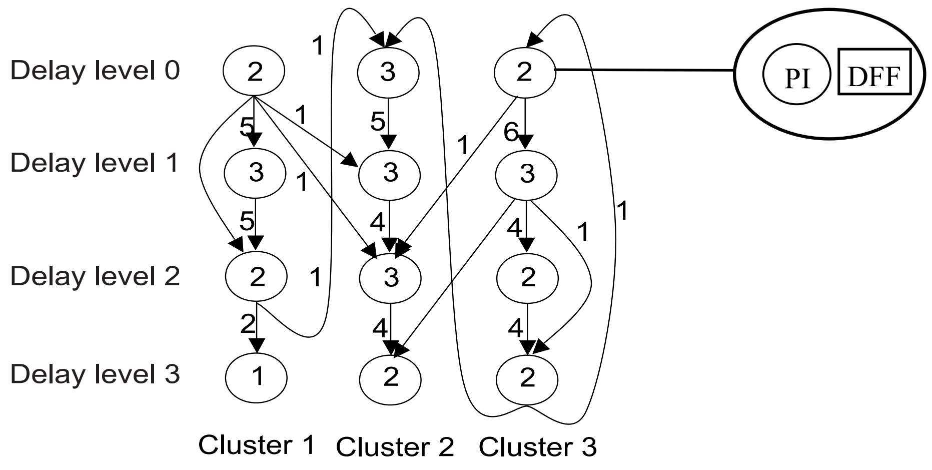

Figure 15 - Completed Delay Structure

```
While there exists latched node to flip-flop connections to be made {
  randomly select a *source_cluster* and *sink_cluster* from the *Latched* matrix
  randomly select *a source_level_node* with unused latched nodes from the
      *source_cluster* using the *Latched Shape*
  *sink_level_node* = 0th delay level in the *sink_cluster*
  make a connection from *source_level_node* to the *sink_level_node*
}
```

Figure 16 – Algorithm to Create the Latched Node to Flip-flop Connections

4.2 Degree Partitioning

After creating the delay structure, the fanout distribution of each cluster (which is a set of fanout values that ultimately will be assigned to each individual node) is partitioned among the level nodes in the cluster. The input into this degree-partitioning step is the delay structure (generated above in Step 1) and the fanout distribution. The output is the delay structure graph with the fanout distribution partitioned among the level nodes such that the sum of the fanout degrees assigned to each level node matches the number of edges that leave the level node. The input and output of this step of the algorithm is illustrated in Figure 17.

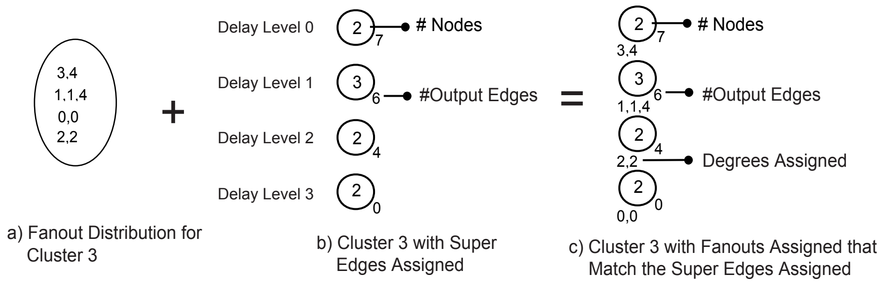
3,4
1,1,4
0,0
2,2
a) Fanout Distribution for Cluster 3
Delay Level 0
Delay Level 1
Delay Level 2
Delay Level 3
2
3
2
2
7
6
4
0
# Nodes
#Output Edges
b) Cluster 3 with Super Edges Assigned
# Nodes
#Output Edges
Degrees Assigned
3,4
1,1,4
2,2
0,0
c) Cluster 3 with Fanouts Assigned that Match the Super Edges Assigned

Figure 17 – Assigning Fanout to Cluster 3 in the Delay Structure

The degree partitioning occurs in three steps. In the first step, an initial assignment is made. In the second step, the degree partition is iteratively improved. In the third step, post processing is done to enforce the equality described above.

The initial fanout assignment in each cluster is made based on the cluster's fanout distribution and its node shape and output shape in the delay structure. It occurs in five steps as given in Figure 18 with Steps 1, 4, and 5 being based on Hutton et al.'s [7] work. Step 1 is performed because nodes with maximum combinational delay are either latched or are primary outputs. Step 2 is performed because latched nodes rarely fanout to more than one node. Step 3 is performed because only latched nodes and primary outputs are allowed to have zero fanout. Step 4 is performed because assigning high fanouts to such level nodes makes it then necessary to assign very low fanouts to these level node which may or may not exist in the number needed.

```
1. Assign each node at delay level $d_{max}$ a fanout of 0.
2. Assign level nodes with latched nodes the lowest unassigned fanout
   degrees
3. Assign any remaining zero degree fanouts remain to level nodes with
   POs.
4. Assign any low fanout degrees to level nodes that have a small output
   degree in relation to the number of its individual nodes
5. Assign the remainder of the fanouts beginning with the largest
   fanouts based on the fanout capacity of the level nodes.
```

Figure 18 – Steps for Initial Fanout Distribution Assignment

After the initial assignment, the degree assignment is improved by swapping fanout degrees between level nodes in an attempt to improve the solution quality defined as:

$$cost_{Degree} = \sum_{LN \in Level\ Nodes} \mathrm{cost}_{Fanout\ Edge\ Misassignment}(LN) + \mathrm{cost}_{Fanout\ Penalty}(LN) \tag{3}$$

Here $cost_{FanoutEdgeMissassignment}$ measures for each level node the absolute difference between the sum of the fanout degrees assigned and the number of output edges that were assigned in Section 4.1 and $cost_{FanoutPenality}$ is a cost that penalizes level nodes with fanouts that will force node violations in the final edge assignment (described below in Section 4.4).

The degrees to be swapped are chosen by first randomly choosing a non-zero fanout degree from the level nodes with higher total fanout than the number of output edges assigned. Next, we randomly choose a non-zero lower fanout degree from the level nodes with lower total fanout than the number of output edges assigned. An example of a degree move is given in Figure 19.

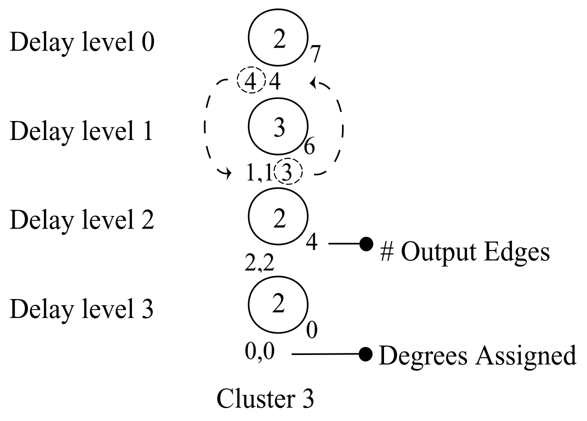

Figure 19 - Example of a Degree Move

We employ an iterative improvement algorithm, in which all changes that improve the cost function are accepted, and bad moves are accepted with a probability of $e^{-\Delta cost}$. Moves are continually generated in the algorithm until there is no change in the lowest cost for 5000 iterations or until the total cost is zero.

After improving the solution quality, some small discrepancies may still exist between the sum of the fanout degrees assigned to each level node and the number of edges that exit the level node. Furthermore, there may still exist level nodes that have fanouts that will force multiple connections between two nodes during final edge assignment. The discrepancies exist in these cases because the output shape of the delay graph structure often does not exactly match the specification of the output shape. These discrepancies are resolved by randomly selecting and decreasing fanout values biasing the degree selection towards larger fanouts or if that cannot be done by adding extra edges. The number of these discrepancies tends to be small involving less than 1% of the total edges.

4.3 Level Node Splitting

The next step in the generation process is to split the level nodes into individual nodes (which will ultimately become 4-input LUTs, primary inputs, and flip-flops in the final generated circuit). The graph is also prepared for final edge assignment. The input into this phase of the algorithm is the fanout-assigned delay graph structure (described above in Section 4.2) with the latched shape, the primary output shape, and the number of primary inputs and flip-flops. The output from this is a graph where the individual nodes have been created at each level node and assigned a logic type from one of flip-flop, primary input, or LUT; where each node has an assigned fanout; where each node has a horizontal position and where all latched nodes and nodes that are primary outputs have been designated. The input and output of this Step for a sample cluster is show in Figure 20. The complete structure with all clusters is called the pre-edge assignment structure and is shown in Figure 21. The algorithm for splitting the level node into individual nodes is given in Figure 22.

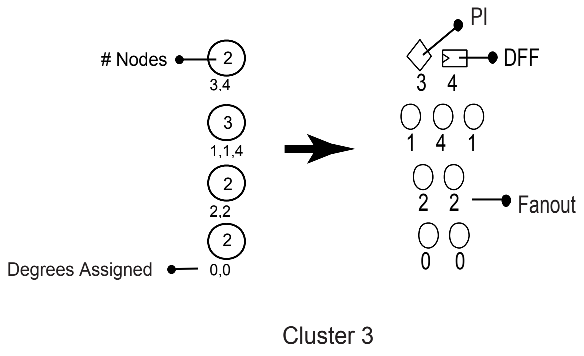

Figure 20 – Level Node Splitting in Cluster 3

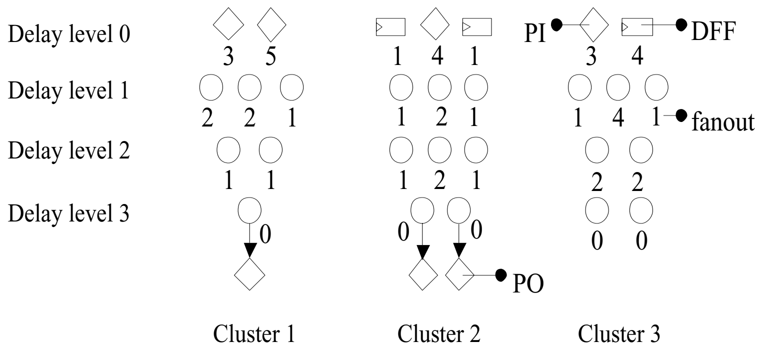

Figure 21 - Pre-edge Assignment Structure

1. Assign each individual node a fanout from the fanouts assigned to its level node.
2. Select nodes to designate as latched by randomly selecting nodes of zero fanout or, if there are more latched nodes than zero fanout nodes, we randomly select nodes of the lowest fanout.
3. Assign a logic type to all nodes.
    a. Designating all nodes at delay levels greater than one as 4-LUTs.
    b. Designate the nodes at the 0th delay level as either a primary input or a flip-flop.
        i. Assign all nodes at the 0th delay level with zero fanout that are not latched to be flip-flops because only flip-flops with primary outputs or primary inputs that are latched can have zero degree fanout assignments.
        ii. Randomly designate the rest of the nodes as primary input or flip-flop until we designate all primary inputs and flip-flops.
4. Assign primary outputs
    a. Attach primary outputs to all non-latched nodes with zero fanout.
    b. Randomly attach any remaining POs to non-latched non-PI nodes.
5. Assign each node a horizontal position to give meaning to the wirelength approximation discussed in Section 3.3.4. Assign according to Hutton et al. [7], in which high fanout nodes are assigned positions that balance them across each level node so as to not skew $\textit{wirelength}_{Approx.}$.

Figure 22 – Splitting the Level Node into Individual Nodes

4.4 Final Edge Assignment

The last stage in the algorithm is final edge assignment. The input into the algorithm is the pre-edge assignment structure described above in Section 4.3. The output from this step is the completed synthetic benchmark circuit. The algorithm proceeds by creating an initial solution and then iterating to improve the solution.

The initial solution is created in two parts. First, we create the connections to the combinational (individual) nodes and secondly we create the connections between the latched nodes and flip-flops.

We make the connections to combinational nodes in the graph by visiting each level node in turn and forming connections to the individual nodes it contains in four steps as given in Figure 23. Our method to construct the initial solution is based on and evolves from Hutton et al.'s work [7].

1. Create a source and destination list
    a. Create the destination node list from the level node's individual nodes
    b. Create the source node list by sampling individual nodes with unassigned fanout from the *source level nodes*, which are all the level nodes with a super edge that inputs into the level node.
2. Ensure that each destination node has its delay level well defined by assigning an edge between it and a source node that is a unit edge length apart.
    a. Choose a source node by randomly sampling the source node list for nodes that are a unit distance apart and choosing the source node that is closest in horizontal position to the destination node. This process of sampling the source nodes for a node that we can make a connection to is called the *edge connection process.*
3. Ensure that the destination nodes are not single-input buffers by assigning them a second edge with the edge connection process. The only restriction on this node selection process is that the destination node cannot make a second connection to the same source node.
4. Form the remainder of the connections.
    a. Randomly select destination nodes and make connections using the edge connection process until either we can no longer make further connections without creating node violations or we run out of source nodes.
    b. If any source nodes remain, randomly select destination nodes and make connections in spite of node violations in the hope that they will be resolved later during iterative improvement.

Figure 23 – Assigning the Individual Edges to Combinational Nodes

After making the connections to the combinational nodes, a similar method is used to make the latched connections. For all of the level nodes at the 0th delay level with flip-flops we assign individual edges as given in Figure 24.

```
1. Create a source node list by sampling unconnected latched nodes from its source level nodes.
2. For all the flip-flops in the destination level node
    a. Randomly sampling the source list a number of times and selecting the latched node that is
       closest in horizontal position to the flip-flop
Create a connection between the latched node and the flip-flop
```

Figure 24 – Assinging the Individual Edges to Flip-flops

After creating the initial solution, we again employ an iterative algorithm that selects certain edges as candidates for relocation and accepts or rejects proposed changes ("moves") based on a cost function. During all moves the algorithm the nodes remain stationary.

The cost function of the algorithm is as follows:

$$cost_{Edge\ Assign} = \beta(\text{wirelength}_{\text{Approx}}(\text{G}) - desired\_wirelength) + (1-\beta)\sum_{n \in V(G)} \text{Number of Violations}(n) \tag{4}$$

Here, *desired wirelength* is a parameter in generation that is discussed more fully in Section 4.5, and *Number of Violations* is a function that returns the number of nodes that have no inputs, too many inputs, two or more connections from the same source node, or if the node is a flip-flop with a connection to itself. The wirelength costs are normalized to the maximum horizontal position multiplied by the number of edges while the *Number of Violations* cost is normalized to the number of edges. We use the factor β to balance the goal of achieving the desired wirelength against the goal of having no node violations. The value is set to 0.02 because the wirelength cost is often much larger than the node violation cost and while achieving the desired wirelength is important it is more important that we have no node violations because they can create sizeable difficulties for our algorithm.

We generate moves 95% of the time purely randomly while 5% of the time we target nodes with violations.

When we generate a purely random move, we start by randomly selecting an edge in the graph. With this edge we attempt one of two possible move types with equal probability. Each move preserves the combinational delay structure and the number of edges that output from each node.

In the first move type, defined as an *edge rotation*, we move the end point of the edge to a new sink node. The new sink node is randomly chosen from within the same level node as the old sink node.

In the second move type, defined as a *double edge swap,* we select a second edge and move the end point of each edge to the other edge's sink node. The second edge is randomly chosen from the edges that output from the same level node that the first edge outputs from.

For the 5% of moves that explicitly attempt to eliminate node violations, we randomly select a node that is in violation. If the violation type is either too many inputs or two or more connections from same source node, we select one of the problem edges and attempt an edge rotation. If the node violation is a flip-flop that connects to itself we attempt a double edge swap where we choose the first edge that connects the flip-flop to itself and the second edge from the list of all edges in the graph that connect to flip-flops and whose choice will preserve the Latched specification.

We again employ an iterative improvement algorithm, in which all valid changes that improve the cost function are accepted, and valid bad moves are accepted with a probability that decreases exponentially with the change in cost. A move is valid if it will not create any flip-flops with loops back to themselves. The algorithm continues until the cost is zero or until the number of moves attempted is a hundred times the number of edges in the graph.

After iteratively improving the graph, if we still have node violations we post process the graph by removing or adding any edges to remove the violations.

4.5 Wirelength Control

To control the post-place and route wirelength of the synthetic circuits that we produce from our generation process we set the *desired wirelength* parameter in the cost function given above using one of three methods. In the first method, *desired wirelength* is set to be the $wirelength_{Approx}$ of the initial solution. In the second method we set *desired wirelength* to be zero to drive $wirelength_{Approx}$ to a minimum while in the third method we set *desired wirelength* to be the value of $wirelength_{Approx}$ measured in the characterization of the original circuit as described in Section 3.3.4.

To evaluate the three different methods of setting $wirelength_{Approx}$ we generated synthetic circuits as described below in Section 5.1, placed and routed them as described in Section 5.2, and examined the post-place and route wirelength.

We found that the *desired_wirelength* conditions that produced the best set of clones were to accept the initial $wirelength_{Approx}$ for combinational circuits and to minimize $wirelength_{Approx}$ for sequential circuits. The reason behind the difference between combinational and sequential circuits can be seen in Figure 25. It shows a typical relationship between the real post-place and route wirelength and the final $wirelength_{Approx}$ for a series of clones generated from two sample combinational and sequential circuits from the MCNC benchmark suite. On both graphs, the real wirelength of the original circuits is marked. We defined this point as the *best desired wirelength point*. This point is on the curve for the combinational circuit and not on the curve for the sequential circuits. This relationship was seen in almost all circuits. Cloned combinational circuits typically used less wirelength than the original circuit at the lowest values of $wirelength_{Approx}$ and more wirelength at the highest levels of $wirelength_{Approx}$. Cloned sequential circuits however, typically used more wirelength at all possible values of $wirelength_{Approx}$. The best desired wirelength point was closest to the initial $\text{wirelength}_{\text{Approx}}$ for combinational circuits while for sequential circuits the best wirelength point was closest to the lowest *wirelength*$_{Approx}$. This suggests that combinational and sequential circuits have very different structures with sequential circuits being more tightly connected.

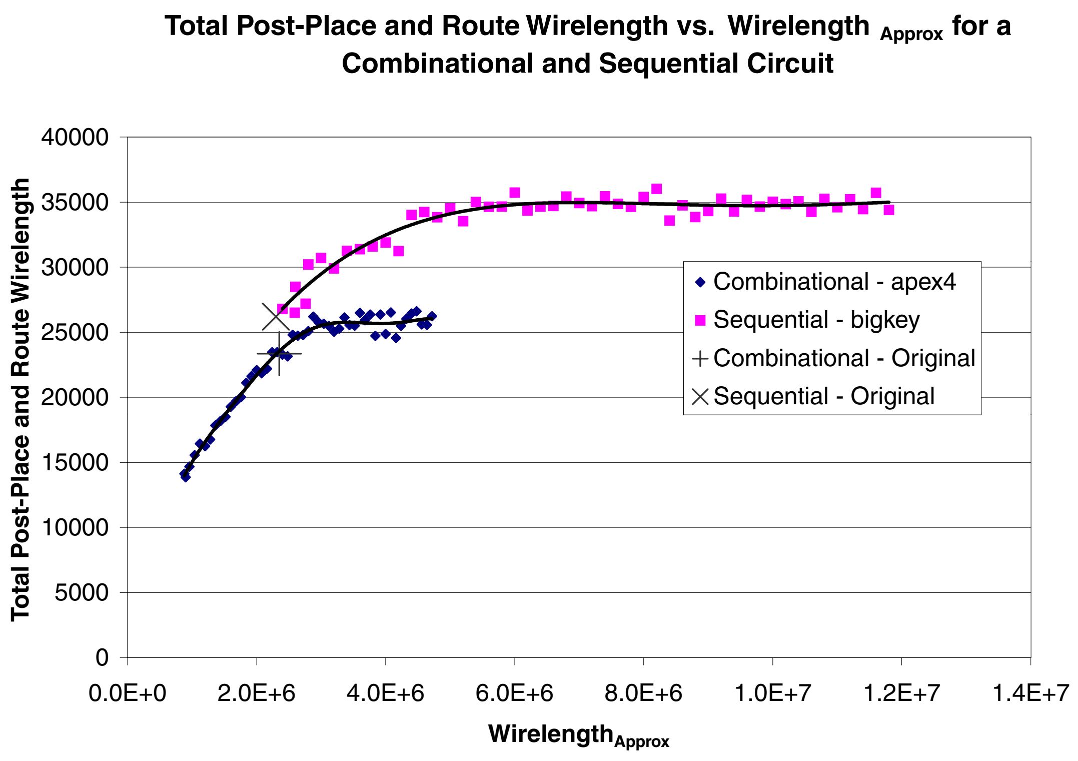

Figure 25 - Wirelength vs. Wirelength$_{Approx}$

4.6 Software Implementation of Generation

We have built an implementation of the algorithm described in this section and called it, *CGen*, after Hutton et al.'s *Gen* [7,8]. CGen takes as input statistics from CCirc and outputs a circuit in BLIF or VHDL format. The source code and executables for CGen can be found at:

http://www.eecg.toronto.edu/~jayar/software/Cgen/Cgen.html

5. VALIDATION

In this section we present measurements of the realism of the synthetically generated circuits. Recall that Hutton et al. [7,8] determined realism by processing the original circuits and synthetic clones of those circuits through the same CAD flow, and comparing measurements on post-place and route results such as total wirelength. We compare on the basis of post-placement and routing wirelength, track count and critical path delay. Our method is a direct validation approach [20]. First, we describe the circuits that we cloned and the parameters that are input into generation. Second, we describe the tool used to do the placement and routing, and the FPGA architecture used to make the measurements. Finally, we present and discuss the results.

5.1 Clone Generation

We used 23 circuits taken from the 17 largest MCNC benchmarks [1] and 6 new circuits created at the University of Toronto. We characterized the circuits with *CCirc* and generated clones with *CGen*.

During characterization, we varied the number of partitions from 1 to 16, in order to study the effect of this parameter. We used both recursive bi-partitioning and *k*-way partitioning types, and used two different node balancing conditions for the two partitioning types.

In generation, for final edge assignment we set *desired wirelength* to the initial $wirelength_{Approx}$ for combinational circuits and zero for sequential circuits, as described in Section 4.5.

For a circuit of 8 clusters, the generation process took, on average, 21 minutes per circuit which is significantly more time than Hutton et al.'s approach [8] but is not prohibitive and could decrease dramatically as software improvements are made.

5.2 Placement and Routing

The circuits were placed and routed with VPR [21,22]. The FPGA architecture targeted was a simple architecture in which logic blocks are a single 4-input LUT and a flip-flop, the wires span only one logic block, and all routing switches are tri-state buffers. The circuits were routed under high stress routing conditions that attempt to find the smallest number of tracks per channel for which the circuits would successfully route.

5.3 Results

The partitioning conditions that produced the best results as measured by placement cost were 8 clusters created through a k-way partitioner that allowed a 20% greater weight in the largest cluster after partitioning.

Using these partitioning conditions, we then measured the average absolute differences between the clone and original circuit for the total-post place and route wirelength, the minimum number of tracks needed to route the FPGA, and the critical path delay as a function of the number of clusters used in characterization. The results are plotted in Figure 26. In this graph we can see that the minimum number of tracks measurement closely follows that of wirelength while critical path delay shows no correlation to the number of clusters. (We suspect that our precise control of delay shape keeps the critical path delay consistently good.)

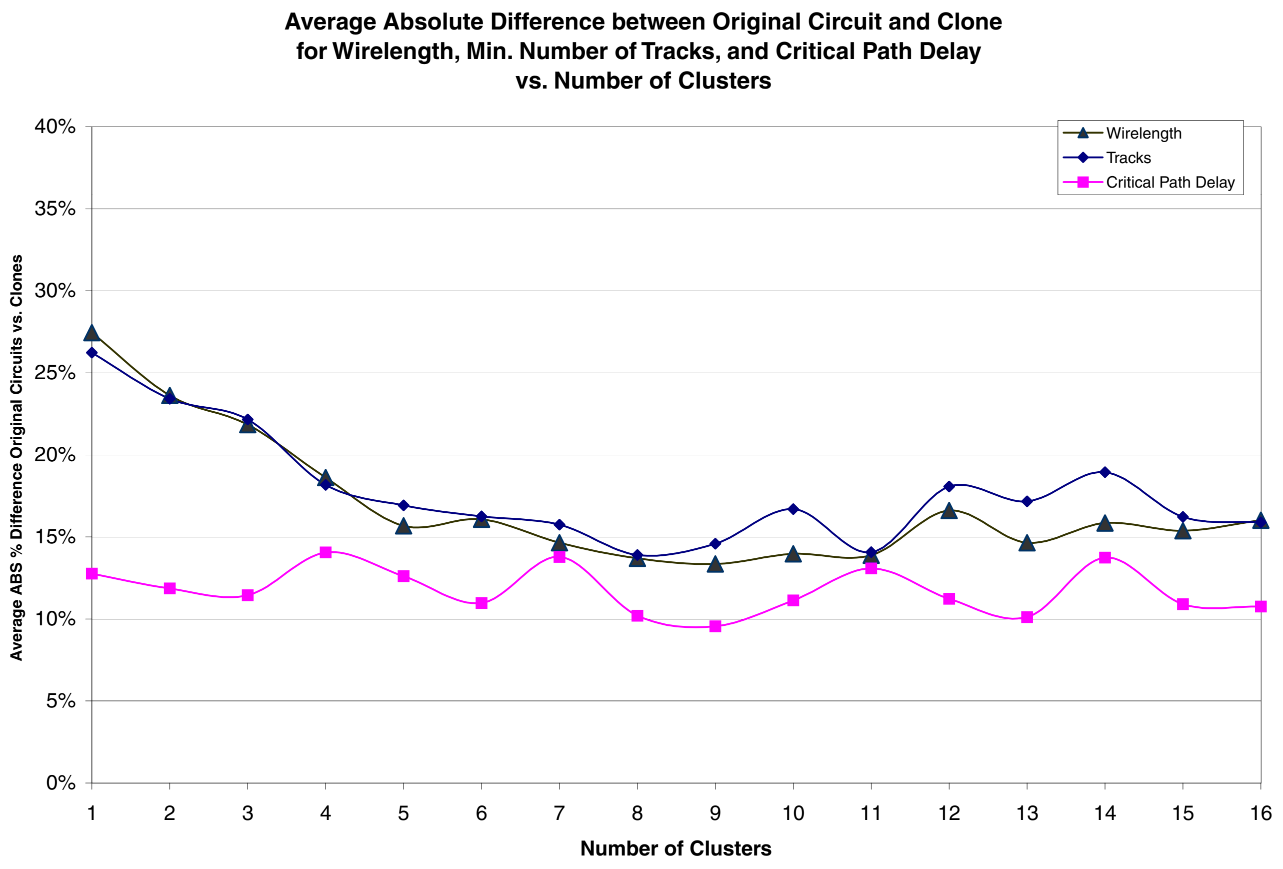

Figure 26 - Wirelength, Minimum Number of Tracks, and Critical Path Delay vs. Number of Clusters

In Table 1 we give measurements of quality for each circuit for 8 clusters. In the table, Orig. denotes the results for the original circuits, MH denotes the results using Hutton et al.'s method [8] to generate clones, and New stands for our new method of generating clones.

As can be seen from Table 1, the mean of the average absolute difference in total detailed wirelength between the clone and original circuit is 14%, which is a significant improvement over Hutton et al.'s method [8] which obtained a difference of 48%. The standard deviation of this mean also has a significant improvement. The absolute difference in the number of routing tracks is 14%, which is also a significant improvement compared to Hutton et al.'s results of 46%. The critical path delay, achieves roughly the same result as Hutton et al.'s work - within about 10% of the original circuit on average. In our judgment, over all three of these measurements, the new synthetic circuits are realistic proxies for the real circuits.

Table 3 - Routability and Critical Path Delay Comparisons between Real and Clone Circuits

		Total Wirelength			Minimum Number of Tracks			Critical Path Delay		
Circuit	Size	Orig	MH % diff	New % diff	Orig	MH % diff	New % diff	Orig	MH % diff	New % diff
alu4	1536	23852	-20%	-8%	14	-21%	-14%	8.4E-08	-18%	8%
apex2	1916	33868	-7%	4%	16	-6%	0%	8.7E-08	-1%	16%
apex4	1270	23059	-16%	-8%	17	-24%	-12%	7.9E-08	-8%	8%
des	1847	31344	49%	22%	10	70%	40%	1.0E-07	-4%	-10%
ex5p	1072	20777	-12%	-14%	17	-12%	-12%	7.1E-08	-3%	1%
ex1010	4608	76083	20%	17%	15	20%	20%	1.6E-07	-16%	-10%
misex3	1411	24100	-13%	-4%	15	-13%	0%	8.6E-08	-19%	-2%
pdc	4591	110830	-13%	12%	23	-22%	13%	2.2E-07	-43%	-39%
seq	1791	33246	-18%	1%	17	-18%	-6%	9.2E-08	-16%	-10%
spla	3706	77666	-1%	18%	22	-14%	0%	1.3E-07	-13%	7%
bigkey	2159	26476	17%	14%	9	78%	22%	5.7E-08	-6%	3%
diffeq	1934	18913	59%	15%	12	42%	0%	7.9E-08	-5%	0%
dsip	1822	23317	-19%	14%	9	22%	11%	7.9E-08	-35%	-11%
elliptic	4854	58017	66%	31%	16	44%	31%	1.1E-07	-13%	-12%
frisc	4444	69611	62%	17%	20	55%	5%	1.4E-07	13%	-5%
s298	1941	25423	8%	0%	11	36%	9%	1.4E-07	-11%	14%
tseng	1482	11520	104%	23%	11	64%	9%	5.5E-08	36%	19%
display_chip	2417	19158	116%	2%	10	90%	0%	7.6E-08	13%	-3%
img_interp	3424	32545	76%	25%	12	58%	17%	9.3E-08	30%	11%
input_chip	1106	7526	83%	11%	8	88%	25%	5.6E-08	16%	8%
peak_chip	1146	6759	60%	4%	9	22%	-11%	8.0E-08	2%	5%
scale125_chip	3856	26834	165%	29%	10	150%	40%	8.8E-08	25%	28%
scale2_chip	1524	11125	99%	19%	9	89%	22%	6.8E-08	15%	5%
abs mean			48%	14%		46%	14%		16%	10%
abs stddev			43%	9%		36%	12%		11%	9%

To ensure that we have not over-tuned our algorithms to the circuits in our test set, we verified our results with a second set of circuits. The second set of circuits consists of fifteen circuits from Sun's Pico Java Processor [26] and two MCNC circuits (s38417 and s38584.1) that were not used in the original test set. We characterized and generated clones of the circuits as in Section 5.1 and place and routed the circuits as in Section 5.2. We then measured the average absolute differences between clone and original circuit for the total-post place and route wirelength, the minimum number of routing tracks needed to route the FPGA, and the critical path delay as a function of the number of clusters used in characterization. The results are plotted in Figure 27. The wirelength and minimum number of routing track results decrease rapidly with the number of clusters until they start to plateau at around 8 clusters, just as with the original test set. The absolute percent difference for these results at 8 clusters is 20% and 21% respectively, which is slightly higher than with our original test set. The critical path delay results are on par with the previous test.

In Table 4 we give measurements of quality for each circuit in the second test set for 8 clusters. In the table, the blanks in the MH columns indicate circuits that caused Hutton et al.'s [8] method to crash. We can see that with our new method the mean and standard deviation of the average absolute difference in total detailed wirelength between the clone and original circuit still shows a significant improvement over Hutton et al.'s method. If we look at the wirelength result for the two MCNC circuits, s38417 and s38584.1, we can see why the average wirelength result for the second test set is slightly higher than for the first. The wirelength result for these two circuits is drastically higher than the result for their original circuits; however, the result is still significantly better than that obtained using Hutton et al.'s method. We do not know why these two circuits use excessive wirelength, however with increasing number of clusters the wirelength difference decreases reaching 71% and 49% respectively with 24 clusters. One possible reason behind the excessive wirelength for these two circuits might be that our method of controlling wirelength is not perfect for all circuits. For almost all other circuits in the second test set, however, the wirelength difference between the clone and original circuit is less than 14%. The average wirelength for these circuits is 11%, which is less than the average found for the first set of test circuits.

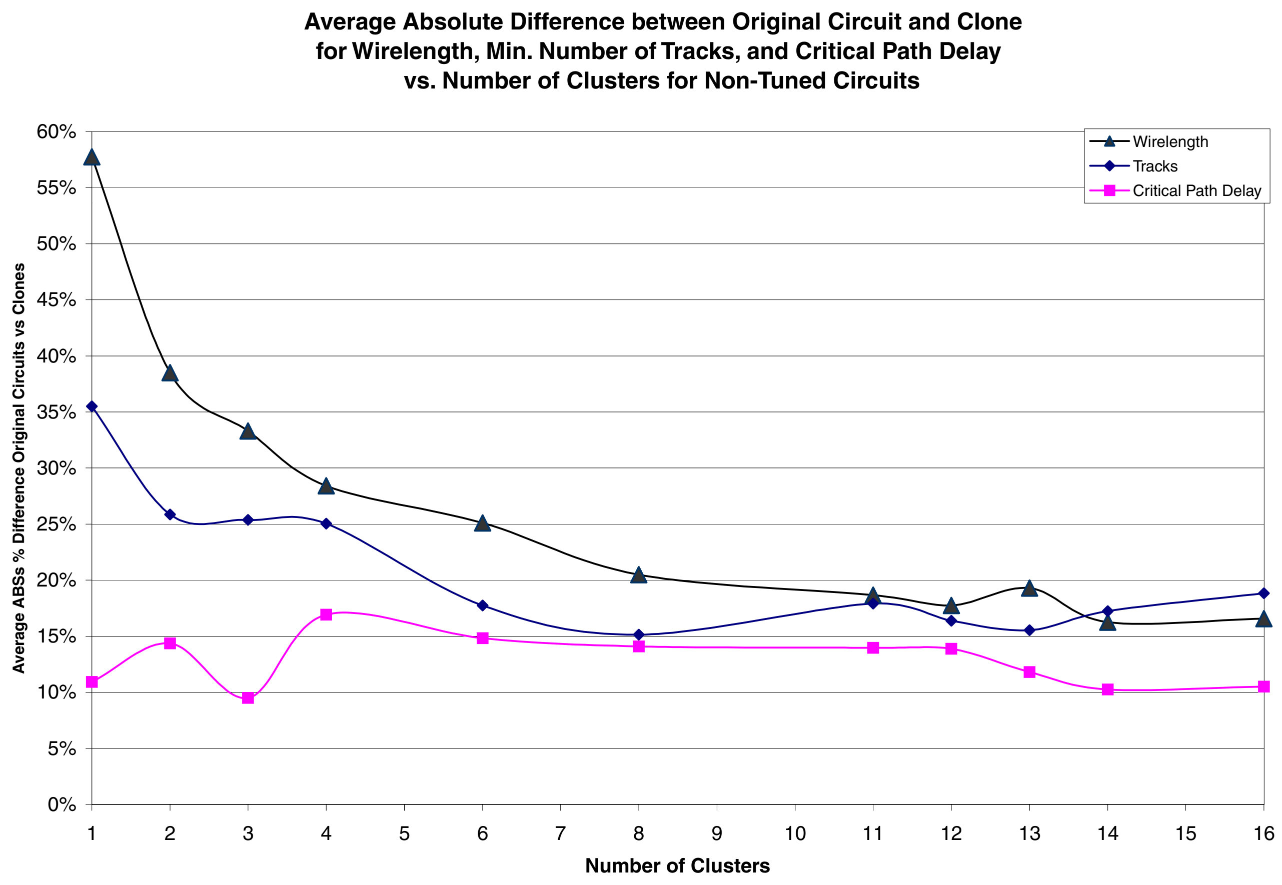

Figure 27 - Wirelength, Min. Number of Tracks, and Critical Path Delay vs. Number of Clusters for Non–Tuned Circuits

Table 4- Comparison between Real and Clone Circuits for Second Test Set

		Total Wirelength			Minimum Number of Tracks			Critical Path Delay		
Circuit	Size	Orig	MH % diff	New % diff	Orig	MH % diff	New % diff	Orig	MH % diff	New % diff
prils_dp	501	4874	12%	8%	11	0%	0%	8.5E-08	28%	24%
rsadd_dp	521	4546	26%	18%	9	0%	47%	1.4E-07	16%	2%
code_seq_dp	588	5110	-29%	19%	8	13%	6%	7.8E-08	-7%	19%
dcu_dpath	1590	23193	62%	6%	19	26%	8%	6.4E-08	22%	4%
ex_dpath	3925	133554	39%	7%	17	59%	0%	2.6E-07	23%	11%
exponent_dp	644	7543	16%	7%	12	25%	8%	8.3E-08	17%	38%
icu_dpath	3813	69064	65%	9%	19	68%	7%	1.3E-07	36%	10%
incmod	1007	11319		1%	12		7%	1.8E-07		3%
imdr_dpath	1410	22384		3%	15		8%	1.8E-07		6%
mantissa_dp	1284	32050		10%	15		13%	6.3E-08		3%
multmod_dp	1893	22532	78%	4%	13	77%	0%	1.3E-07	10%	47%
pipe_dpath	798	7351	4%	0%	8	63%	19%	5.2E-08	8%	0%
smu_dpath	866	10023		10%	9		33%	1.5E-07		14%
ucode_dat	1806	43554		9%	16		82%	8.8E-08		13%
ucode_reg	230	799	86%	48%	3	133%	0%	2.5E-08	-36%	0%
s38417	7465	71038	201%	100%	11	200%	33%	8.6E-08	23%	3%
s38584.1	7489	69751		90%	11		82%	8.8E-08		47%
abs mean			56%	20%		60%	21%		20%	14%
abs stddev			55%	30%		61%	27%		11%	16%

We conclude from the second test set results that the basic generation parameters are not over-tuned.

In the results presented so far, we have grouped the clones by the number of clusters used in their partitioning. However, the best number of clusters to partition a circuit into, as defined by wirelength, is not constant across all circuits. For each circuit there exists a best and most natural number of clusters to partition a circuit into. In Table 5, we give measurements of quality for each circuit in both the first and second test set for the number of clusters that produces the best wirelength results. As can be seen from this table, the mean of the average absolute difference in total detailed wirelength between the clone and original circuit is 9%, The absolute difference in the number of routing tracks is 12%. The critical path delay, achieves roughly the same result as Hutton et al.'s work [8] - within about 12% of the original circuit on average. If we remove circuits s38417 and s38584.1 from consideration, those numbers drop to 6% for wirelength, 10% for minimum number of routing tracks, and 10% for critical path delay respectively.

We include this comparison because selecting the most appropriate number of clusters on a per circuit basis could be considered a valid part of the characterization process, albeit one that is more labour intensive.

Table 5 - Comparison between Real and Clone Circuits for Best Number of Clusters

			Total Wirelength			Tracks			Critical Path Delay		
Circuit	Size	Clusters	Orig	MH	New	Orig	MH	New	Orig	MH	New
				% diff	% diff		% diff	% diff		% diff	% diff
alu4	1536	13	23852	-20%	-1%	14	-21%	-7%	8E-08	-1%	11%
apex2	1916	3	33868	-7%	1%	16	-6%	-6%	9E-08	-1%	17%
apex4	1270	13	23059	-16%	0%	17	-24%	0%	8E-08	-8%	0%
des	1847	16	31344	49%	16%	10	70%	10%	1E-07	-4%	-11%
ex5p	1072	12	20777	-12%	-3%	17	-12%	-6%	7E-08	-3%	3%
ex1010	4608	8	76083	20%	17%	15	20%	20%	2E-07	-16%	-10%
misex3	1411	9	24100	-13%	-1%	15	-13%	0%	9E-08	-19%	-6%
pdc	4591	3	110830	-13%	9%	23	-22%	4%	2E-07	-43%	-36%
seq	1791	11	33246	-18%	0%	17	-18%	0%	9E-08	-16%	-6%
spla	3706	5	77666	-1%	11%	22	-14%	0%	1E-07	-13%	-1%
bigkey	2159	13	26476	17%	5%	9	78%	11%	6E-08	-6%	5%
diffeq	1934	16	18913	59%	7%	12	42%	8%	8E-08	-5%	-4%
dsip	1822	13	23317	-19%	9%	9	22%	11%	8E-08	-35%	-32%
elliptic	4854	9	58017	66%	20%	16	44%	19%	1E-07	-13%	-7%
frisc	4444	8	69611	62%	17%	20	55%	5%	1E-07	13%	-5%
s298	1941	8	25423	8%	0%	11	36%	9%	1E-07	-11%	14%
tseng	1482	9	11520	104%	20%	11	64%	27%	6E-08	36%	16%
display_chip	2417	8	19158	116%	2%	10	90%	0%	8E-08	13%	-3%
img_interp	3424	15	32545	76%	10%	12	58%	0%	9E-08	30%	2%
input_chip	1106	9	7526	83%	5%	8	88%	13%	6E-08	16%	-1%
peak_chip	1146	11	6759	60%	0%	9	22%	0%	8E-08	2%	-6%
scale125_chip	3856	13	26834	165%	25%	10	150%	50%	9E-08	25%	16%
scale2_chip	1524	5	11125	99%	13%	9	89%	11%	7E-08	15%	12%
prils_dp	501	3	4874	12%	0%	11	0%	-18%	8.5E-08	28%	3%
rsadd_dp	521	11	4546	26%	6%	9	0%	0%	1.4E-07	16%	0%
code_seq_dp	588	16	5110	-29%	-1%	8	13%	50%	7.8E-08	-7%	-14%
dcu_dpath	1590	14	23193	62%	1%	19	26%	0%	6.4E-08	22%	20%
ex_dpath	3925	13	133554	39%	2%	17	59%	-6%	2.6E-07	23%	24%
exponent_dp	644	6	7543	16%	-1%	12	25%	0%	8.3E-08	17%	24%
icu_dpath	3813	12	69064	65%	-2%	19	68%	-5%	1.3E-07	36%	6%
incmod	1007	8	11319		1%	12		8%	1.8E-07		19%
imdr_dpath	1410	8	22384		3%	15		7%	1.8E-07		-4%
mantissa_dp	1284	16	32050		-4%	15		0%	6.3E-08		2%
multmod_dp	1893	2	22532	78%	3%	13	77%	-8%	1.3E-07	10%	35%
pipe_dpath	798	8	7351	4%	0%	8	63%	13%	5.2E-08	8%	10%
smu_dpath	866	11	10023		9%	9		11%	1.5E-07		-23%
ucode_dat	1806	11	43554		-2%	16		-13%	8.8E-08		4%
ucode_reg	230	14	799	86%	7%	3	133%	33%	2.5E-08	-36%	-2%
s38417	7465	18	71038	201%	71%	11	200%	73%	8.6E-08	23%	40%
s38584.1	7489	18	69751		49%	11		36%	8.8E-08		14%
abs mean				51%	9%		51%	12%		17%	12%
abs stddev				47%	14%		45%	16%		11%	11%

6. CONCLUSIONS

We have introduced a new circuit characterization and new synthetic generation techniques that significantly improve the quality of synthetic circuits over previous methods.

The new characterization captures a hierarchy of a circuit by partitioning the circuit into clusters and characterizing each cluster and the connections between the clusters.

Our new generation techniques impose this hierarchy on the synthetic circuits. The generation techniques use iteration to tightly control the generation process in contrast to all known synthetic circuit generators that use constructive approaches.

The new synthetic generation techniques were judged realistic by cloning real circuits and comparing clones on the basis of post place and route statistics. For 8 clusters, the real and clone circuits differed by 14% for total detailed wirelength, 14% for minimum number of tracks needed to route each circuit, and 10% for critical path delay. If we further choose the best number of clusters for each circuit (as defined by wirelength) we found that the real and clone circuits differed by 9% for total detailed wirelength, 10% for minimum number of tracks needed to route each circuit, and 10% for critical path delay.

This is a key (but not final) step towards the goal of having the ability to create larger circuits than already exist.

Several areas are open to future research. The key next step is to see how circuit structures combine and scale with size so that we can generate larger circuits. A second area is to explore alternatives to using $wirelength_{Approx}$ to control wirelength during synthetic circuit generation. One such alternative would be to place the circuit during final edge assignment on an FPGA to obtain positions for the individual nodes in the graph and then use this information to obtain the precise wirelength information for edges. A third area is to examine the use of different partitioners in characterization to see what effect they have on synthetic circuit quality. A fourth area of research is to prove the realism of synthetic circuits by showing that their substitution for real circuits does affect the conclusion of any FPGA architecture or CAD tool algorithm experiment.

7. ACKNOWLEDGEMENTS

The authors wish to thank NSERC for financial support, Mike Hutton for his valuable help and advice, Vaughn Betz for use of VPR, and George Karypis and Vipin Kumar for use of hMetis.

8. REFERENCES

[1] S. Yang, “Logic Synthesis and Optimization Benchmarks, Version 3.0,” Tech. Report Microelectronics Centre of North Carolina. P.O. Box 12889, Research Triangle Park, NC 27709 USA, 1991.

[2] C. J. Alpert, "The ISPD98 Circuit Benchmark Suite", Proceedings International Symposium on Physical Design”, pp. 85-90, Apr. 1998.

[3] ITC'02 SOC Test Benchmarks,http://www.extra.research.philips.com/itc02socbenchm/#Benchmarks

[4] Altera Inc., http://www.altera.com/products/devices/stratix/stx-index.jsp, 2003.

[5] Xilinx Inc., “Virtex-II Pro Platform FPGA Handbook v2.0,” Oct. 2002.

[6] SEMATECH, “International Technology Roadmap for Semiconductors”, 2001, pp. 39-40. Available at: http://public.itrs.net/Files/2001ITRS/Home.htm

[7] M.D. Hutton, J. Rose, J.P. Grossman, and D. Corneil, “Characterization and parameterized generation of synthetic combinational circuits,” IEEE Trans. CAD, vol. 17, no. 10, pp. 985-996, Oct. 1998.

[8] M.D. Hutton, J. Rose and D. Corneil, “Generation of Synthetic Sequential Benchmark Circuits,” In IEEE Trans. CAD, vol. 21, no. 8, pp. 928-940, Aug. 2002.

[9] J. Darnauer and W. Dai, “A Method for Generating Random Circuits and its Application to Routability Measurement,” In 4th ACM/SIGDA Int'l Symp. on FPGAs, FPGA96, pp. 66-72, Feb. 1996.

[10] B. S. Landman and R. L. Russo, “On a Pin Versus Block Re- lationship for Partitions of Logic Graphs,” IEEE Trans. Comp., C-20(12) pp. 1469-1479, 1971.

[11] K. Iwama and K. Hino, “Random Generation of Test Instances for Logic Optimizers,” In Proc. 31st Design Automation Conference, pp. 430-434, 1994.

[12] K. Iwama, K. Hino, H. Kurokawa, and S. Sawada, “Random Benchmark Circuits with Controlled Attributes.” In Proc. 1997 European Design and Test Conference, 1997.

[13] J. E. Harlow III and F. Brglez, “Synthesis of ESI Equivalence Class Combinational Circuit Mutants” Technical Report 1997-TR@CBL-07-Harlow, CBL, CS Dept., NCSU, Box 7550, Raleigh, NC 27695, 1997

[14] D. Ghosh, N. Kapur, J. E. Harlow III, F. Brglez. "Synthesis of Wiring Signature-Invariant Equivalence Class Circuit Mutants and Applications to Benchmarking," In Proceedings, Design Automation and Test in Europe, pp. 663, Feb. 1998.

[15] D. Ghosh, F. Brglez and M. Stallmann, “Generation of Tightly Controlled Circuit Classes for Problems in Physical Design", Report from CBL, CS Dept., NCS. Available at: *http://www.cbl.ncsu.edu/\-publications/\-\#2000-TR@CBL-01-Ghosh*

[16] J. Pistorius, E. Legai, and M. Minoux, "PartGen: A Generator of Very Large Circuits to Benchmark the Partitioning of FPGAs," IEEE Trans. CAD., vol. 19, no. 11, pp. 1314-1321, 2000.

[17] S.J.E. Wilton, J. Rose, Z.G. Vranesic, “Structural Analysis and Generation of Digital Circuits with Memory”, IEEE Transactions on VLSI Systems, vol. 9, no. 1, pp.223-226, Feb. 2001

[18] D. Stroobandt, P. Verplaetse, and J. Van Campenhout, "Generating Synthetic Benchmark Circuits for Evaluating CAD Tools," IEEE Trans. CAD., 19, no. 9, 1011-1022, Sept. 2000.

[19] P. Verplaetse, D. Stroobandt, and J. Van Campenhout, "Synthetic Benchmark Circuits for Timing-driven Physical Design Applications", Proc. of the Intl. Conf. on VLSI, pp. 31-37, June 2002.

[20] D. Stroobandt, J. Van Campenhout, and P. Verplaetse, "On Synthetic Benchmark Generation Methods," Proc. IEEE Intl. Symp. On Circuits and Systems, vol. IV, pp. 213-216, May 2000.

[21] V. Betz and J. Rose, "VPR: A new packing, placement and routing tool for FPGA research," in 7th Int. Conf. On Field-Programmable Logic, pp.213-222, 1997.

[22] V. Betz, J. Rose and A. Marquardt, "Architecture and CAD for Deep-Submicron FPGAs," Norwell, MA, USA, Kluwer Academic Publishers, 1999.

[23] P.D. Kundarewich, "Synthetic Circuit Generation Using Clustering and Iteration," M.A.Sc. Thesis, University of Toronto, 2002.

[24] Michael Hutton, "Characterization and Automatic Generation of Benchmark Circuits" Ph.D. Thesis, University of Toronto, 1997.

[25] G. Karypis and V. Kumar, "Multilevel k-way Hypergraph Partitioning", Proc. ACM/IEEE Design Automation Conf., pp. 343-348, 1999.

[26] PicoJava Microprocessor Core, Sun Microsystems, http://www.sun.com/microelectronics/pico

Karti Mayaram
Editor-in Chief
IEEE Transactions on CAD

Dear Dr. Mayaram,

Thank you for arranging the review of our paper, "Synthetic Circuit Generation Using Clustering and Iteration". We have now revised the paper extensively as per the comments from the reviewers, and below we respond to all of the comments from the reviewers. The original comments are preceded by the bar ("|") character, followed by our response.

Paul Kundarewich and Jonathan Rose

Review Number 1.

| Although the general motivation and the basic ideas of the current methodology
| are well-presented, a major concern is the readability of the (lower-level)
| technical content (Sections 3 and 4). In my opinion, the style of presentation
| makes the paper accessible to a very narrow audience -- only those researchers
| very familiar with Hutton's work and other works in the field.
| It is true that the authors give an overview of Hutton's work (Section 2.1)
| and the other relevant contributions in the field (Section 2.2). However,
| the technical aspects in Sections 3 and 4 are quite difficult to follow
| in the absence of, e.g., an illustrative example (which is badly needed,
| in my opinion). The authors try to exemplify from time to time:
| e.g., the creation of the combinational (Fig. 9) and sequential (Fig. 10)
| delay structures, but the related comments are quite insufficient.
| I think an illustrative example accompanying the presentation would
| significantly improve the readability.

We have made a number of changes in order to address the readability of Sections 3 and 4.

In Section 3, we have added more concrete examples to provided a better description of the new characterizations introduced in that section. These can be found in Tables 1 and 2, and Figures 9 and 10.

In Section 4, we have added an illustration of the generation process with an example. The example is generated from the characterizations of Figure 6 in Section 3. This example appears in Figures 12, 17, and 20. They show in graphical form the basic input and output of the various steps in the algorithm. This will help provide the reader with a more global view of what is going on in each step.
To reduce the amount of text that the reader is presented with we have moved the descriptions of the algorithms into Figures 14, 16,18, and 22-24. To simplify the descriptions they have been put into pseudo-code format.

With the increased number of figures and the reduced amount of main text we hope that these sections have become more readable.

| Section 5 presents many experiments which show clear improvements over
| Hutton's work. However, this is the only work the current results are compared
| with. Would it be possible to present some comparisons with other existent
| works presented in Section 2.2 ? Or at least explain why other comparisons

| are not possible.

This is a very good point. It would be very beneficial if it was possible to make these kinds of comparisons. The field isn't as mature as, say the partitioning, placement and routing fields, where this kind of direct comparison has been made possible through extensive standardization of benchmarks. In synthetic circuit generation, the basic goal is to achieve "realistic" circuits, but there is not agreed-upon quantitative metric of what realistic means. Indeed, Hutton first proposed that realism meant being able to synthesize a clone with similar wirelength to the original. Ghosh and Brglez assume that they have something realistic by mutating an existing design. There are no standard input and output formats of these generators which would permit a direct comparison. We do note in Section 2.2 *qualitative* rather than quantitative differences between our approach and others:

a) Clearly would not realistic (Darnauer and Dai, Iwama)

b) Have different goals (Ghosh/Brglez, Wilton, Pistorius)

We outline these reasons in Section 2.2 where I make the comments:

Darnauer and Dai
"However, the approach lacks control over the fanout and unit delay profile of gates in the circuit."

Ghosh/Brglez:
"However, the approach does not lend itself to the possibility of scaling the mutants to larger circuit sizes, which is a key motivation behind synthetic circuit generation."

Iwama:
"While the resulting circuit is realistic from a logical standpoint, the method suffers from a lack of control over the physical properties of the netlist...No validation was done on any of the synthetic circuits as the application of their work was towards evaluating the ability of synthesis tools to reoptimize circuits they transform."

Pistorius:
"Success is not judged on the basis of the wirelength or delay properties and it is unclear as to whether under these latter criteria the circuits would prove realistic. No characterization or method to judge success is given at the second level of hierarchy."

Wilton
"The weakness of the method is its use of only combinational MCNC benchmarks in logic portions of the generated circuits."

Stroobandt/Verplaetse
"This method produces circuits that are too regular and that have unrealistic delay profiles [19]. Verplaetse et al. [19] attempt to fix these problems and achieves good wirelength results with the real circuit and clones differing by 7% on average. It is unclear whether or not the delay profile problem has been sufficiently fixed since no direct comparison is made between the synthetic circuit delay profile and that of a real circuit."

Specifically for Stroobandt/Verplaetse's work we do not make comparisons because:

1. Their software does not output circuits in a format that the t-vpack/VPR flow can currently handle.
2. While their synthetic circuit generation software is on the web we have not found software that generates their input files. Therefore, we would need to hand generate these files. These files require that the Rent's Rule exponent be known at different circuit sizes. We are not aware of any existing tool to do this. The time to calculate these parameters for each MCNC circuit we felt was prohibitive. Furthermore, although they do provide a few sample input files, these samples are very small and possibly do not represent the complexity of their generation efforts. Thus, we are not sure if such a comparison would be fair if one were indeed to be made.

| The authors offer links to the source code and executables
| (Sections 3.4 and 4.6), which is very good.

Thank you!

| Minor remarks

| 1. Please check ref.[18] in your Reference list: I think there is an error at the title.

This has been corrected.

| 2. You are often writing a comma between subject and predicate (which is
| not grammatically correct). E.g., ``Darnauer and Dai [..], generate ...''
| You are doing this mistake several times in Section 2.2 .

This has been corrected.

Review Number 2.

Comments to the Author

|Your model that incorporates clustering as well as the idea of equating flip flops to primary inputs
|as being the top most level is quite clever. Your iterative algorithm also allows for expandability in
|modelling new and different parameters which is quite good. Some specific comments are made
|below.

| Sec 2.1.2 - You mention "several other parameters are omitted here for simplicity". You need to
| at least list them here. Otherwise, the reader is left wondering about the validity of your
| omission.

These parameters have now been listed.

| Section 3.3.2 - You mention that the matrices are weakly correlated but there is no proof for that.

We have added a reference to Paul Kundarewich's thesis, which is available online, which gives the proof for this. There is too much data to include in the paper.

| Section 4 - General comment: There is too much prose in the document when a straightforward
| pseudo code would significantly enhance the comprehension of the ideas in this paper.

Agreed. As mentioned in the comments to reviewer #1 the amount of prose has been reduced by changing the description of the code to pseudo-code and moving it into figures.

| You really mean structural VHDL right? You should be specific.

This has been corrected.

| Section 4.1 - This is dire need of a simple figure otherwise one has to plod through the prose
| laboriously to understand what is going on here.

As mentioned above to reviewer #1 we have added a figure that shows the input and output of this step of the algorithm.

| Figure 9 is extremely confusing - you need to mark the nodes differently (cannot figure out which
| is the weight, which is the #inputs, #outputs etc.)

We have simplified the figure and added comments to explain the characteristics.

| Section 4.1.1 - In the cost equation, how do you convert the matrix into a scalar (Cost comb)?
| There are multiple ways of doing this - you should be precise in mentioning here.

We have added more detail to describe the calculation.

| General comment: in several places you mention "hill climbing iterator ...". This is not only
| repetitious but confusing since it makes the reader wonder if these simulated annealing
| problems are fundamentally different or are just the cost functions different.

We have replaced this term with “iterative improvement algorithm.” The algorithm is a modified form of an annealer.

| General comment: Your equations need to numbered.

This has been corrected.

| Section 5 - The second sentence: what does "comparing measurements on the results" mean?
| Isn't that exactly what you are doing too?

This has been corrected to “comparing measurements on post-place and route results such as total wirelength”

| In the results, you don't talk about the "scalability" of this approach since generation of bigger
| circuits than current circuits is the goal of this work as explicitly stated in the introduction.

The reviewer is correct in that this is our goal. But before we achieve our goal of scaling circuits to larger sizes we need to first get circuits of the same size correct. We mention this in the introduction. To further reinforce this we have added a statement to the conclusion.

| Also, on page 36, you mention "We do not know why" the two circuits have excessive
| wirelength. Even a possible conjecture is better than leaving the issue completely unresolved.

We have added the conjecture “One possible reason behind the excessive wirelength for these two circuits might be that our method of controlling wirelength is not perfect for all circuits.”

| In general, I like the ideas in the paper. However, the writing needs to be polished up. The
| authors need to describe more of the rationale for why they ended up making the decisions that
| they did. There also needs to be a big picture description of the algorithm. The algorithm
| descriptions feel like they are prose descriptions of the program and it is difficult as a reader to
| keep all the issues in mind when reading.

This has been done. As stated above a lot of the prose has been taken out and put into pseudo-code sections. A rational for breaking things up has been provided – essentially that the whole optimization problem is too computationally difficult and complex.

Review Number 3.

| You have significantly improved the work by Mike Hutton and the
| results section is devoted to saying how much better your method is
| than Hutton's work. It is a pity though that you did not compare
| your results to the last available methods from other
| groups. Especially the work by Verplaetse (for which you say you do
| not know if the delay profile problem is solved) seems to be worth
| comparing with (he has a web site with the code available on it).

See the comments above to Reviewer #2 on this issue.

| in your assessment of the quality of benchmarks (sections 1, 2.1.4,
| and 5.3), you could use the terminology introduced by Verplaetse et
| al. in [Proceedings of the International Conference on VLSI,
| 2002. pp. 31-37] about direct and indirect validation.

We have added a section in the background and later stating that our approach is a "direct" one according to their terminology, vs. indirect.

| The background and previous work section is elaborate and contains
| all other related works that I know of with a relevant description
| of their merits and drawbacks. I appreciate this in your paper. Just
| a thought: would it not be more logical to start with section 2.2
| and then, based on the reasoning that Hutton's approach is better
| suited to your needs, continue with section 2.1?

This is a good suggestion, thank you. We have changed it to be this way.

| I have problems understanding the meaning of the first sentence of
| section 3. Any clustering method induces some form of hierarchy (by
| the way in which the clusters are chosen). Introducing hierarchy
| simply by using clustering is straightforward (nothing new). It is
| the way in which you choose the clusters (and hence defines the
| hierarchy) that brings something new.

The reviewer makes a good comment. We are not trying to say that we've invented hierarchy, simply adding it to the large suite of characterizations that Hutton et al started created.

| The "new" model in section 3.1 may be new to Hutton's model, but is
| the most natural way to describe sequential circuits. We have been
| working with this model (implicitly) for a long time now. So I would
| not stress the novelty of this too much.

We agree, but would argue there is novelty in the *addition* of hierarchy to all of the other characterizations.

| Also, in section 3.2, I
| fail to see why the hierarchy in figure 6 would resemble the
| "natural" hierarchy. What makes you state this? I would argue that a
| random clustering approach will NOT find the natural hierarchy in a
| circuit at all!

This is a good observation – we cannot claim to be finding the natural hierarchy, which we agree is a difficult problem. We will change this to say find simply "a hierarchy."

| Section 4: I would still call your method constructive, even with
| the iteration. The iteration just ensures that the constructive
| approach can be adjusted subsequently to improve upon the
| results. So the first step is still constructive, the subsequent
| steps might be called mutations. In my view, your approach combines
| "cloning" and "mutations".

The reviewer is correct that the must be a constructive initial phase. This is true for all iterative solutions.

| It is generally bad practice to have long subscripts (as in the
| equations on pages 21, 25 and 29). Subscripts should be at most 3
| characters and explained in the corresponding text.

We agree that this deviates from convention, but would prefer to retain the long subscripts so that we don't have to add more words to the paper.

| wonder if it is not possible to prevent violations from the
| beginning instead of having to remove them afterwards (end of
| section 4.1.1). Have you looked at this?

Yes, in fact that is what is attempted when the initial solution is created.
In almost all cases it is not possible to prevent node violations and that is why we need to iterate.

| I did not understand the reasons behind the "desired_wirelength"
| conditions as explained on page 31 and in figure 13. Figure 13 is
| supposed to explain the reasons but (1) it shows a typical example
| (this gives no reason, it just shows it might be a good approach)

The purpose of the "desired_wirelength" parameter is to control the wirelength in the final placed and routed circuit. We have changed the first sentence to emphasize this fact.

What was previously: "We have three methods of setting the *desired wirelength* parameter in the cost function given above."

Is now changed to: “To control the post-place and route wirelength of the synthetic circuits that we produce from our generation process we set the *desired wirelength* parameter in the cost function given above using one of three methods.

We took an experimental approach to evaluating the effectiveness of the 3 approaches to setting desired_wirelength. Figure 13 is trying to explain why the 1st approach (*desired wirelength* is set to be the *wirelengthApprox* of the initial solution) works well for combinational circuits while the 2nd approach (set *desired wirelength* to be zero to drive *wirelengthApprox* to a minimum) works well for sequential circuits. We state this when we write:

“To evaluate the three different methods of setting $wirelength_{Approx}$ we generated synthetic circuits as described below in Section 5.1, placed and routed them as described in Section 5.2, and examined the post-place and route wirelength.
We found that the *desired_wirelength* conditions that produced the best set of clones were to accept the initial $wirelength_{Approx}$ for combinational circuits and to minimize $wirelength_{Approx}$ for sequential circuits.”

After explaining the figure (which we rewrite below to improve the clarity) we suggest a reason why this is so when we state that “This suggests that combinational and sequential circuits have very different structures with sequential circuits being more tightly connected.”

| and (2) it is not clear from the paper what is shown in the
| figure. I would have expected measured wire lengths in function of
| time (or number of iterations) for several values of
| desired_wire length. Please explain this figure (and the section)
| much more clearly.

What was previously: “It shows a typical relationship between real wirelength and $wirelength_{Approx}$ for two example combinational and sequential circuits from the MCNC benchmark suite. On both graphs, the point at which the real wirelength of the clone matches or would have matched the real wirelength of the original circuit is marked. We defined this point as the *best desired wirelength point*.”

Is now: “It shows a typical relationship between the real post-place and route wirelength and the final $wirelength_{Approx}$ for a series of clones generated from two sample combinational and sequential circuits from the MCNC benchmark suite. On both graphs, the real wirelength of the original circuits is marked. We defined this point on the graph as the *best desired wirelength point*.”

| Also, the words "within the graph" and "not
| within the graph" on page 31, line -8, should probably be "on the
| curve" and "not on the curve" since all points are definitely within
| the graph (entire figure).

This has been corrected.

|Your results section should also contain figures of the time needed
| to obtain the results. I can imagine that the iterative techniques
| (hill climbing approach) take a lot more time than Hutton's original
| approach (especially important since you are aiming at large

| circuits). But how much is the difference? Is the time increase
| prohibitively expensive?

This is a good point, we have added the following paragraph:

"For a circuit of 8 clusters, the generation process took, on average, 21 minutes per circuit which is significantly more time than Hutton et al.'s approach [8] but is not prohibitive and could decrease dramatically as software improvements are made."

| At the end of page 40, you present a fourth area of future research
| and should be the first on your list of things to do. It would even
| be better if you could include such results in this paper.

We agree that this is an important area for future work.

| Language issues:

| abstract, last two sentences: write this in present instead of past
| tense

This has been corrected.

| page 4, line -2: "its delay" -> "their delay"

This has been corrected.

| page 6, line 1: "For a circuit, Hutton" (insert ",")

This has been corrected.

| when referring to paper [7], you always refer to "Hutton" alone
| although there are multiple authors. Therefore it would be better to
| refer to "Hutton et al.". Then, you can also use "they" instead of
| "he" as you already do on the first line of page 8.

This has been corrected.
All Hutton have been changed to Hutton et al.
All Hutton's have been changed to Hutton et al.'s

| page 8, line -6: "are partitioned" instead of "are partitioning"

This has been corrected.

| in section 2.2, remove the comma's each time after "<names>
| [<ref>]," in the beginning of a sentence.

This has been corrected.

| paragraph 3.3.4, line 2: "in characterization that we will use" What
| do you mean?

| page 25, line 5: remove comma in "by first, randomly choosing"

This has been corrected.

| page 28, line 5: "numbers of connections": remove first "s"

This has been corrected.

| page 28, line 8: "a unit distance" instead of "a unit distant"

This has been corrected.

| page 36, line 12: "." at end of sentence

This has been corrected.

| page 36, line -5: "results ... is" -> "result ... is"

This has been corrected.

| page 36, line -5: "than (for) the first."

This has been corrected.

| page 36, line -4: "than (the results for) their original circuits;"

This has been corrected.

| page 38, line 1: "We conclude that from...": remove "that"

This has been corrected.

| section 6, line 1: "techniques": remove "s"

We think the plural is more appropriate.

| section 6, line 2: "improve(s)"

Same as above.

| page 40, line 9: "If we further, choose": remove ","

This has been corrected.