Characterization and Parameterized Random Generation of Digital Circuits

Michael Hutton†, J.P. Grossman‡, Jonathan Rose ††, and Derek Corneil†
Departments of Computer Science†, Mathematics‡ and Electrical and Computer Engineering††
University of Toronto, Ontario M5S 1A4
{mdhutton@cs, jp@eecg, jayar@eecg, dgc@cs}.toronto.edu

Abstract

The development of new Field-Programmed, Mask-Programmed and Laser-Programmed Gate Array architectures is hampered by the lack of realistic test circuits that exercise both the architectures and their automatic placement and routing algorithms. In this paper, we present a method and a tool for generating parameterized and realistic random circuits. To obtain the realism, we propose a set of graph-theoretic characteristics that describe a physical netlist, and have built a tool that can measure these characteristics on existing circuits. The generation tool uses the characteristics as constraints in the random circuit generation. To validate the quality of the generated netlists, parameters that are not specified in the generation are compared with those of real circuits, and with those of "random" graphs.

1 Introduction

There is a need for benchmark netlists in order to compare and test the quality of new ASIC architectures and physical design algorithms. However, useful benchmarks are rare—they are usually too small to effectively test large future-generation products, and those large enough are often proprietary.

Some attempts to alleviate this problem have been the MCNC benchmarks [10], the PREP benchmarks [8], and the use of random graphs. The use of random graphs is appealing because the supply is infinite, and the circuit size can be specified. However, only a small subset of random graphs can be considered reasonable with respect to electrical constraints (such as gate fanin or fanout), topological properties (such as maximum delay) and packaging constraints such as the number of pins. Compared to random graphs, circuits are inherently tame for implementation in gate arrays, and exhibit hierarchical structure which leads to empirical observations such as Rent's Rule[1] [7].

In independent work, Darnauer and Dai [3] have proposed a method of generating random undirected graphs to meet a given I/O ratio and Rent parameter [3]. Their work reports results only for small circuits (from 77 to 128 lookup-tables) so it is difficult to evaluate the method's success at achieving the target parameterization because few partition points are available to determine the Rent parameter, r.

[1]Rent's Rule: For a "reasonable" partition of a circuit into at least 5 modules, the relationship between the average number P of terminals/pins on a module, and the average size B of a module follows the relationship $P = k * B^r$, where k is a constant and r is the *Rent parameter* which is a characteristic of the circuit in question.

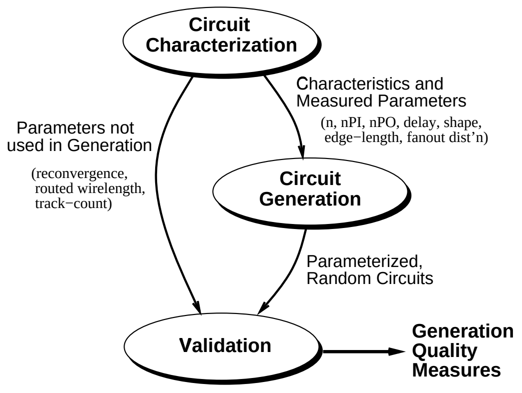

Circuit Characterization
Characteristics and Measured Parameters
(n, nPI, nPO, delay, shape, edge−length, fanout dist'n)
Parameters not used in Generation
(reconvergence, routed wirelength, track−count)
Circuit Generation
Parameterized, Random Circuits
Validation
Generation Quality Measures

Figure 1: Approach to Circuit Generation

In this paper we propose a method to generate benchmark circuits, illustrated in Figure 1, which combines the advantages of random graph generation and the realism obtained by using actual circuits. We have defined a set of graph-theoretic characteristics and parameters of circuits and measured these on real circuits up to 4500 LUTs to form a *profile* of realistic circuits. This measurement is done with a new software tool called CIRC. The profile is used by a second tool, GEN, to generate a random circuit with the specified parameters. In this way we produce netlists with the characteristics of realistic circuits, but also have the freedom to vary crucial parameters, including the size of the generated circuit. The interaction between the analysis and generation tools is of fundamental importance: CIRC can be used to analyze any private collection of circuits and determine alternative profiles for input to GEN. In this paper we will validate the quality of the generated circuits by measuring circuit characteristics that are not specified in the generation, as illustrated in Figure 1.

This system allows for features not possible with standard benchmark sets. For example, one parameter can vary while others are fixed or scaled appropriately, to generate a "family" of circuits.

In this paper we define the new algorithmic problem of random circuit generation with constraints. The circuit generation problem is very difficult and we present a heuristic algorithm to solve it inexactly, in a program called GEN.

The paper is organized as follows: Section 2 outlines the characterizations of circuits used for generation and validation of the random circuits and Section 3 describes the generation algorithm. We measure and discuss the quality of the generated circuits in Section 4 and then conclude.

2 Circuit Characterization

This section describes some of the statistical and structural characteristics of circuits which we have identified. For the purposes of this paper we focus on combinational circuits only, and have used the MCNC benchmark circuits to form the

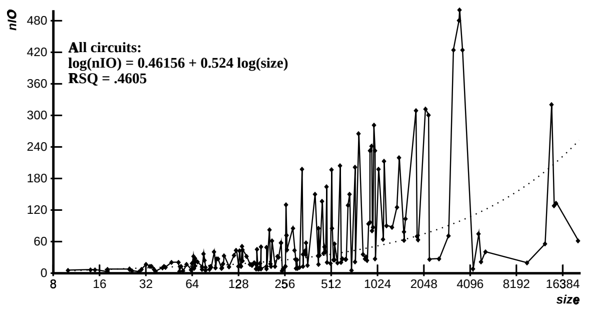


Figure 2: Size (2-LUTs) vs. I/O for MCNC circuits.

basis for characterization and parameterization. Note that the users of our system could profile their own circuits with CIRC and specify the results as parameters to GEN (or modify the program default file) to customize the types of circuits generated.

2.1 Pre-processing of Analyzed Circuits.

The MCNC benchmark circuits were converted from EDIF to BLIF, optimized with SIS [9] (keeping the better result of script.rugged and script.algebraic) then technology mapped using FLOWMAP [2] into k-input lookup tables. Specifically, each circuit was mapped 7 times, into 2-input LUTs, 3-input LUTs up to 8-input LUTs. We chose to use lookup-tables because of their simplicity, functional completeness and the ease of changing to different LUT-sizes. We believe that the structural differences of circuits are sufficiently captured by the use of LUTs to determine valid characterizations.

2.2 Characteristics and Parameters.

There are two different types of characterizations: those needed to determine reasonable defaults for generation parameters which the user does not specify and those which characterize the fundamental structure of a circuit. In the remainder of this section we propose a set of characteristics.

1. Circuit Size and Number of I/Os. The most basic characteristic of a circuit is the relationship between the size of the circuit (number of LUTs, n) and the number of primary inputs (nPI) and outputs (nPO). (Define $nIO := nPI + nPO$.) Using linear regression and experimentation, we have determined that a Rent-like functional relationship, $\log(nIO) = a + b \cdot \log(n)$ best captures the relationship between IOs and circuit size[2]. A simple linear relationship best describes the division of I/Os between inputs and outputs: $nPI = c + d \cdot nPO$. Figure 2 shows a plot of $\log(n)$ vs. $\log(nIO)$, and a least-squares regression line for the Rent-like relationship. We note that simply determining values for the coefficients a, b, c, and d does not capture the increase in variance with n so we model these coefficients as Gaussian distributions around the best-fit line. The actual equations are shown in Figure 10.

2. Combinational Delay. Define $d(x)$, the *delay* of node x, as the maximum length over all directed paths beginning at a PI and terminating at x, corresponding to the unit delay model. The delay, $d(C)$ (or just d), of a circuit is the maximum delay over all nodes in C. Using a similar empirical analysis to the above, we have determined the relationship between delay and circuit size (n).

[2]Note that Rent's Rule explicitly does not apply uniformly for the circuit as a whole (i.e. to predict I/O given n), so we use different functional forms for ranges of n, determined empirically. The actual relationship is a piecewise combination. See Figure 10.

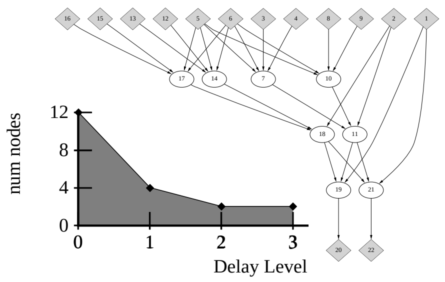


Figure 3: Shape function.

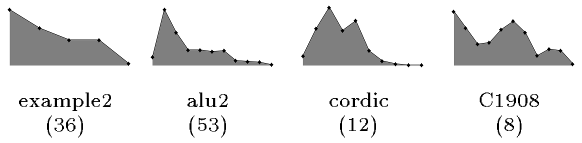


Figure 4: Different shape functions.

3. Circuit Shape. Combinational delay is very important in the characterization of circuits, precisely because it is so important in the design and synthesis process. Define the *shape function*, shape(C), of a circuit as the vector of the number of nodes at each combinational delay level. Figure 3 shows a small example circuit (cm151a), and its shape function (12, 4, 2, 2) displayed as a histogram. Note that we do not count primary outputs in the shape function. While these examples are mapped to 4-LUTs, the form of the function changes only slightly with the LUT-size.

The interesting thing about shape is that most circuits tend to have similar shapes. Figure 4 shows four shape functions. Of the 109 combinational multilevel circuits in the MCNC set, 36 have a shape which is strictly decreasing from the primary inputs (as "example2"), 53 have a "conical" shape, fanning out from the inputs to an extreme point, then strictly decreasing (as "alu2"), 12 have the conical shape with a "bump" and only 8 did not fit into these categories. Importantly, this is fundamentally different from degree-constrained graphs we generated randomly, which had much "flatter" shapes.

4. Edge Length Distribution. Since nodes have a well-defined delay, we can define the length of a directed edge by length$(x, y) = d(y) - d(x)$. Clearly, the edge length is always between 1 and delay(C), and the *edge length distribution* is well defined. In the example of Figure 3 there are 24 edges of length 1, and 2 each of length 2 and 3, so the edge length distribution is (24, 2, 2, 0). We find that almost all circuits have an edge-length distribution with a similar shape: a large number of edges of length 1, and a quickly falling distribution over the combinational delay of the circuit.

5. Fanout Distribution. Define fanout(x) as the number of edges leaving a node x. A circuit's *fanout distribution* (the number of nodes with fanout 0, 1, 2, etc.) is an important structural parameter. Note that fanin is less interesting for technology-mapped circuits because they have an *a priori* constraint on fanin. We have determined the fanout distributions of the MCNC circuits, and have developed an algorithm [5] which generates reasonable fanout distributions given the above size and shape parameters.

6. Reconvergence. Reconvergence occurs when multiple fanouts from a single node x in the circuit branch back to-

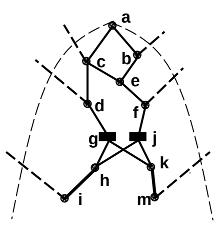


Figure 5: Reconvergence in combinational circuits.

gether at a later point y - we say the circuit is *reconvergent at y*. Many circuits exhibit reconvergent fanout, but in widely varied amounts, and so an appropriate characterization is to quantify this amount.

Define the *out-cone* of a node x (in a circuit with no directed cycles) to be the recursive fanout of x: all nodes reachable by a directed path from x. Figure 5 shows out-cone(a).

Define the *reconvergence number of node x*, $R(x)$, as the ratio of the number of fanin-2 (i.e. "reconvergent") nodes in out-cone(x) to the size of out-cone(x). (Note: for simplicity here we define reconvergence for 2-LUT mapped circuits only[3].) In the example, $R(a) = 3/12$ or 0.25. In most sequential circuits, the existence of flip-flops[4] could result in *back edges* and directed cycles, and $R(a)$ would no longer be well-defined both because the definition of out-cone is insufficient and because the natural extension of counting reconvergent nodes results in overcounting. A more generalized definition of reconvergence, based on spanning out-trees rather than a simple ratio, takes this into account [5] but it is beyond the scope of this paper. The reconvergence number of an entire circuit is the weighted average (by cone size) of the reconvergence number of the circuit's primary inputs.

The reconvergence numbers of the MCNC circuits vary between 0.0 and 0.92 (the theoretical maximum for 2-LUT mapped circuits is 1.0), with a relatively even distribution of circuits through the range 0.0 to 0.85. R is somewhat a measure of complexity of the logic—we find intuitively simple logical functions have low R, and more complex functions have higher R. Combinational and sequential arithmetic circuits fall mostly in the range 0.0 to 0.5, whereas finite state machines are mostly in the range 0.6 to 0.85.

In a physical sense, there is some correlation between R and the shape of a circuit. Using the examples of Figure 4, circuits which have an exaggerated conical shape, such as alu2 ($R = .53$) and cordic ($R = 0.45$) tend to have higher reconvergence values, whereas circuits like example2 ($R = 0.17$) are lower. This also tends to explain the difference between combinational and sequential circuits because the first "sequential level" of most finite state machines tends to be very conical, due to a low I/O to logic ratio.

3 Circuit Generation

Figure 6 shows an example output from GEN for the parameterization: n=23, k=2, nPI=7, nPO=2, d=5, shape=(.38, .31, .19, .12), fanouts=(.09, .65, .13, .04, .09), edges=(.75, .25).

[3] R for circuits mapped to 4-LUTs is calculated by $R(x) = \frac{\sum_{v \in \text{out-cone}(x)} \log_2 \text{fanin}(v)}{|\text{out-cone}(x)|}$ rather than the simple ratio. Thus $0 \leq R \leq \log_2(k)$. A still more complicated definition is required for sequential circuits—see [5].

[4] Nodes g and j are flip-flops, but they do not have edges which form a directed cycle.

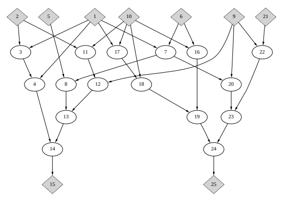


Figure 6: Example of a completely parameterized circuit.

The GEN program consists of two functional stages. The first is to determine a complete parameterization of the circuit to be generated, using partially-specified user parameters and default distributions. The second stage is to output a random circuit with that parameterization, which we deal with first.

3.1 The Generation Algorithm.

The inputs to GEN are n, nPI, nPO, (or nIO), d (depth), k (LUT-size), the shape function, and the edge length and fanout distributions. The output is a netlist of k-input lookup-tables. Reconvergence is not a generation parameter but we use the reconvergence number of generated circuits in the validation process of Section 4.

The input shape function is scaled[5] to determine the set N ($|N| = n$) of all nodes, partitioned into sets N_i ($i = 0..d$) of nodes at each combinational delay level. The fanout distribution is scaled with n to become the set D of available out-degrees, where d_j ($j = 1..n$) represents the eventual fanout of one node j. This also determines the number of edges m in the circuit. Finally, the edge length distribution is scaled to create the set E of available edges, where length(e_h) ($h = 1..m$) represents the length of edge h. This gives rise to the following combinatorial assignment problem, illustrated in Figure 7.

Circuit Generation Problem

Given: D, N_i, E.

Find: assignments of nodes in N to each $d_j \in D$, and pairs of nodes for each $e_h \in E$ such that:

I The number of edges leaving any $x \in N$ is exactly its corresponding fanout d_x.

II All $x \in N_i$ have at least one fanin from N_{i-1} ($i>0$). (i.e. d(x) equals its assignment.)

III Fanin(x) $\leq k$ for all $x \in N$.

IV Fanins of $x \in N$ are distinct (i.e. no two fanouts of gate y are both inputs to x.)

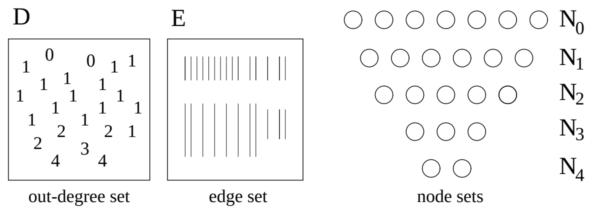


Figure 7: The generation/construction problem.

This assignment problem appears to be computationally

[5] First it is scaled to length d by linear interpolation, then adjusted to ensure there are sufficient nodes or POs to absorb the minimum fanout of each level.

difficult and we speculate it is NP-hard. It is important, moreover, to have a nearly linear time algorithm in order to generate large circuits. Therefore we solve the problem heuristically.

Our approach is to first determine, in steps 1 and 2, an assignment of edges and out-degrees to delay levels and then, in step 3, to divide the levels into individual nodes. We begin by collapsing the nodes at each delay level into a single "level-node," n_i, labeled with $s_i = |N_i|$, the number of nodes it represents. This produces a modified problem: Assign level-nodes for each d_j and pairs of level-nodes for each e_h such that: (I') the number of edges leaving any n_i is the sum of the s_i fanouts d_j assigned to it. (II') There are at least s_i fanins to n_i from n_{i-1} $(i > 0)$. (III') $\text{Fanin}(n_i) \leq k \cdot s_i$. (Note: there is no IV'.)

Step 1: Edge Assignment To Levels. To assign edges to level-nodes, we first meet constraint II', by assigning s_i unit edges (length one) between n_i and n_{i+1} $(i = 0..d-1)$. We then iteratively assign non-unit edges to meet constraints on minimum fanout and fanin of each n_i. This is based on the interaction between shape and the supply of edges of each length, which restrict where edges of certain lengths will need to go in order to find a feasible solution[6]. Once as many edges as possible have been assigned in this way, we assign the remaining non-unit edges probabilistically: we calculate a distribution representing the ability of pairs of level-nodes to support edges of each length (based on their available fanin—constraint III'—and fanout), and assign the edges randomly by sampling this distribution. Note that the remaining unit edges are not assigned until after the next step.

Step 2: Fanout Assignment: The construction to this point is illustrated in the left side of Figure 8 (although the remaining unit edges are already shown). The fanout of each n_i, $(\text{fanout}(n_i))$ can be approximated from the already assigned edges. We now assign the elements of D to levels so as to satisfy the number of edges emanating from each level, and use the remaining unassigned unit edges to ensure a solution which exactly assigns all of the $d_j \in D$ and meets I' without violating III'.

Define $G_i = \{g_{ij}\}$ to be the (initially empty) set of assigned fanouts (from D) for level i, and the *unsatisfied fanout* of level i, $u_i = \text{fanout}(n_i) - \sum_j g_{ij}$. Define the *mean unsatisfied fanout* of level i, $\bar{u}_i = u_i/(s_i - |G_i|)$.

The partition of out-degrees is done in a greedy manner[7]. At each stage we assign the largest element of D to the n_i with the largest mean unsatisfied fanout $\bar{u}_i$, then update G_i, u_i and $\bar{u}_i$. Unit edges are assigned when u_i falls below the number of remaining nodes $(s_i - |G_i|)$. The process repeats until all d_j have been assigned (meeting I'), completing the situation illustrated in Figure 8.

Step 3: Final Edge Assignment. We now return to the original problem specification. It remains to assign the edges and fanouts currently associated with n_i to individual nodes of N_i. We proceed from the top down. At each level, we assign a fanout value g_{ij} to each one of the s_i nodes of N_i and randomly connect g_{ij} edges from the fanout list of n_i to

[6]This is one of the more involved steps in the algorithm, and it is not possible to describe all the details here. More detail will appear in a Tech. Report, and the code is is publically available from the authors.

[7]We have we have omitted some details in the algorithm wich are necessary to ensure that a reasonable solution can be found without backtracking.

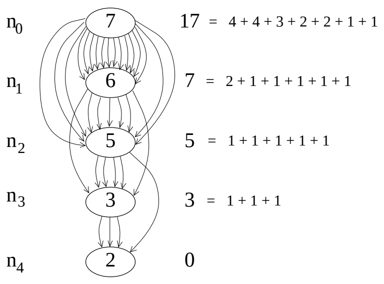


Figure 8: The generation algorithm

it. The connection of fanin nodes is more difficult, because we have to complete the assignment without violating constraints II, III or IV. First we satisfy II by connecting one unit edge to each $x \in N_i$. Then we build a list of edges from previous levels which have to be connected, and a list of (already defined) heads of these edges. Proceeding iteratively with the list of head-nodes, sorted by the fanout to n_i of that node, we assign the remaining edges. The assignment is random, but biased to enforce a certain amount of "locality" (closeness) in the result. Specifically, when choosing the sets of fanout nodes for each node at a level, we sample l times (empirically $l = 6$) to minimize a distance metric between node labels in the fanin and fanout sets. Although we have not yet succeeded in producing a better characterization of locality, it is a fundamental feature of circuits, and so it is important to introduce a degree of locality in the generation process.

The final result of the generation algorithm on the structure of Figure 8 is the original example of Figure 6.

3.2 Closeness of Fit.

Steps 1 and 2 are not guaranteed to find a valid solution; when necessary, GEN will modify D or create extra nodes to allow successful completion. With extensive experimentation, we have determined five versions of Step 1 which together deal with the characteristics of different input sets, and we take the best result of each with Step 2 as input to Step 3. To evaluate the success of the heuristics, we compared the MCNC circuits to circuits generated with their exact profiles—GEN typically made only minor changes to D or created few extra nodes, and these only on relatively few circuits[8]. The degree of modification is larger when GEN uses the entire default set—yet 85% of generated circuits are reasonable immediately, meaning they have a feasible number of I/Os.

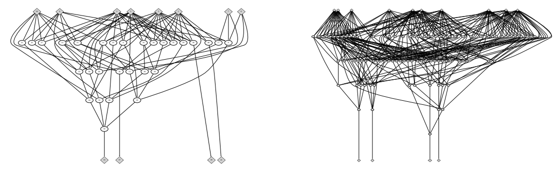
3.3 Examples.

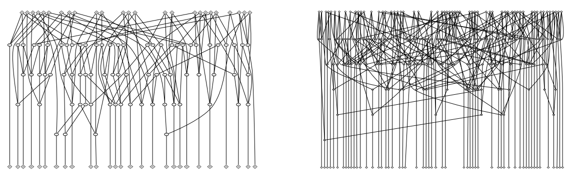
Figure 9(c) shows four different circuits produced by GEN using the default parameter distributions. We note that these circuits appear to be "normal" circuits, and include many features such as areas of high-fanout. The visual "quality" of the circuits is most striking when one observes the similarity to MCNC circuits (Figure 9(a)) and the contrast between MCNC circuits and random graphs (Figure 9(b)).

3.4 Parameterization.

GEN is augmented with a sophisticated C-like language, SYMPLE, for parameter generation. The default distributions are written in this language, and the user can specify modifi-

[8]We find d and nPI correspond exactly always, nPO exactly on about 75% of circuits, and on most others of by a reasonable amount. Shape, edge-length and out-degree distributions were modified less than 25% of the time, and usually by a minor amount.


(a) MCNC circuits sqrt8 and sao2


(b) Random 4-regular digraphs

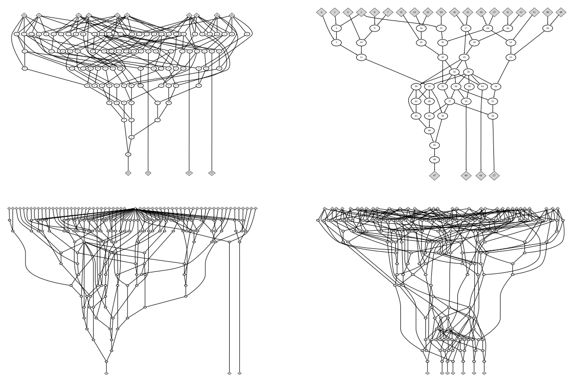

(c) Varied circuits produced by GEN.

Figure 9: MCNC circuits vs. random graphs vs. GEN.

```
smallIO = gauss(0.431*size2 + 1.448, 1.562;
medIO   = gauss(0.225*size2 + 8.026, 8.006);
largeIO = size2*gauss(-0.054*log(size2)+0.54,0.148);
defaultIO = rand(20, 500);
nIOfl =   size2<30 ? smallIO : size2<100 ? medIO :
          size2<1000 ? largeIO : defaultIO;
 nIO  = min(n-depth+2, nint(nIOfl));
```

Figure 10: The SYMPLE language: nIO specified by a Gaussian distribution around a Rent-like relationship (GEN default).

cations on the command line, or in a script. SYMPLE provides a great deal of control over parameters. For example, nIO is currently defined as per Figure 10 in the default script.

SYMPLE allows parameters to be specified as constants, drawn from statistical distributions, chosen as functions of other parameters, or as a range of parameters (in which case multiple circuits are output). The latter case is illustrated in Figure 11, where the circuit size varies from 60 to 100 by 20: SYMPLE scales related parameters (e.g. depth and shape) yet retains the similarity of other properties[9]. This ability to scale circuits while retaining fundamental similarities introduces an entirely new paradigm for evaluating the scalability of architectures and algorithms.

[9]This is because the coefficients of unspecified relationships between parameters are drawn first, then scaled to the different circuits, rather than chosen independently.

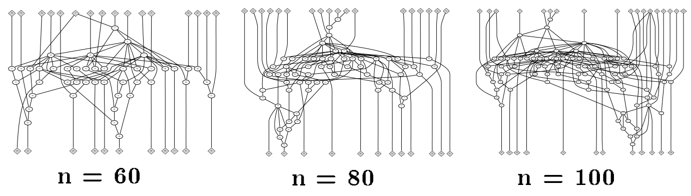


Figure 11: A GEN circuit family ({`k=2; n=60..100#20`}).

4 Validation

In this section we judge the quality of the generated circuits with respect to parameters not specified in generation: reconvergence, and post-placement and routing wirelength and track count. Since one of the primary applications of the circuits produced by GEN is to test and evaluate physical design algorithms, the point of this exercise is to determine how reasonable the output is for this process.

We constructed the exact profile of the MCNC circuits (i.e. n, nPI, nPO, d, shape, fanout and edge length distributions), and generated corresponding circuits meeting those profiles with GEN. Our method of validation is to compare the unspecified parameters of the MCNC circuits against those of the corresponding generated circuits and against “random graphs” of the same size.

Because the exact definition of a random graph varies, we now have to be precise: the most common usage of the term refers to a graph $G(n,p)$ on n vertices with each possible edge existing with equal probability p. However, this is so drastically unlike a real circuit ($G(n,p)$ would be hopeless to route for even small p) that we have found it a more reasonable comparison to use a random k-regular graph—a random graph such that each node x has fanin(x)+fanout(x)=k—as these graphs are more realistic in an electrical sense and are relatively easy to generate. We will compare against circuits mapped to 4-LUTs, and so we will use, for each circuit, the appropriate $k \in \{4,5,6,7\}$ to generate approximately the same number of edges. Two drawbacks of this method are that random k-regular graphs have an inordinate number of I/Os (approximately 20% of nodes) and no high fanout nodes, but they provide a convenient comparison to non-parameterized random generation.

4.1 Reconvergence.

Reconvergence, R, is not a parameter to GEN. Reconvergence captures numerous properties of a circuit, including high fanout, and the interaction between shape, edge length and fanout distribution, all of which affect the ability to place and route the circuit. We calculated R for the generated circuits and compared them to those of the original circuits from which the generation profiles were extracted and to those of random graphs of the same size. The results for the largest 12 MCNC circuits, and means for these 12 and all circuits, are shown in Table 1. Note that $0 \le R \le 2$ for 4-LUT mapped circuits. (See the footnote on page 3.)

We found that, for 70% of generated circuits, R was within 0.1 of the value for the corresponding MCNC circuit. This indicates that the correlation for a crucial parameter R did carry through the generation process. The mean for each class is shown in the table.

In contrast, the reconvergence numbers of the random graphs did not match the MCNC circuits at all. We observe that random graphs exhibit diminishing R as n increases. This is due to the two factors mentioned earlier: the absence of high-fanout nodes and the large number of I/Os.

4.2 Routability.

To test the "routability" of our output circuits, we used a locally available tool to place and global route the sets of MCNC circuits, generated circuits, and random graphs described above. The circuits are compared on two different metrics: the number of tracks per channel required to successfully route, and the total wirelength of the global routing.

The software which we used [1] chooses a minimal square grid to support the size of the circuit, and minimizes both maximum track-count per channel and total wirelength.

Table 2 shows the statistics for the largest 12 MCNC circuits and the summary statistics (differences) for these and all 114 combinational circuits. We see that the track count for the generated circuits differed by 11%, on average, from the corresponding MCNC circuit, whereas the random graphs differed by 63%. Wirelength differed by 11% for the generated circuits and 78% for random graphs. For the largest 12 circuits, the generated circuits used slightly more wirelength and tracks than the MCNC circuits, but the random graphs used much more. These results clearly show the circuits produced by GEN are very similar to the MCNC originals and significantly more realistic than random graphs as benchmark circuits.

		R		
	size	mcnc	gen	rand
C7552	945	0.49	0.45	0.05
ex5p	1072	1.12	1.22	0.27
i10	1252	0.72	0.48	0.09
apex4	1270	0.90	1.17	0.23
misex3	1411	0.55	0.72	0.24
alu4	1536	0.50	0.62	0.22
seq	1791	0.48	0.48	0.21
des	1847	0.50	0.30	0.07
apex2	1916	0.47	0.59	0.20
spla	3706	0.97	1.08	0.13
pdc	4591	1.01	1.13	0.10
ex1010	4608	1.08	1.09	0.10
(12)		0.73	0.79	0.17
(114)		0.39	0.43	0.29

Table 1: Reconvergence: MCNC vs. GEN vs. random.

		Tracks			Wirelength		
	size	mcnc	gen	rand	mcnc	gen	rand
C7552	945	5	8	13	5705	8320	15918
ex5p	1072	10	10	21	14748	13309	27904
i10	1252	7	9	19	10427	13155	28738
apex4	1270	9	9	23	16653	14567	34423
misex3	1411	8	10	24	16780	16912	40152
alu4	1536	8	7	26	16434	15556	45177
seq	1791	9	9	27	22111	21590	57040
des	1847	7	9	23	16229	24116	50294
apex2	1916	9	10	29	23556	25072	63418
spla	3706	10	13	19	49676	62043	167832
pdc	4591	13	14	19	73875	81982	225679
ex1010	4608	8	19	28	55002	92530	231655
(12)		(diff)	28%	218%	(diff)	22%	181%
(114)		(diff)	11%	63%	(diff)	11%	78%

Table 2: Routability: MCNC vs. GEN vs. random.

5 Concluding Remarks

In this paper we have introduced a new method for generating realistic parameterized benchmark circuits. The circuit generation is derived from the measurement of several new graph-theoretic properties which we propose in this paper. As a result the circuits are much more realistic than random graphs. It has been shown that the quality of the circuits (as measured by reconvergence and routability) is comparable to an existing benchmark set and much better than that of random graphs that don't use these properties. Because of the close tie between characterization and generation, users are able to characterize their own circuits using CIRC and create defaults which more closely meet their own needs (rather than the MCNC defaults).

Using this method, we can generate a large set of circuits with the properties of the largest MCNC benchmark circuits. It remains to be seen if even larger circuits (which could easily be generated) have realistic circuit behaviour.

The GEN algorithm is fast, requiring less than 1 minute of SUN Sparc4 time to produce a circuit with 30000 4-LUT nodes. The code is publically available from the authors.

In the future we will expand the GEN system to generate sequential circuits (with flip-flops, back-edges and cycles) and to join sub-circuits together hierarchically. We are currently working on the ability to generate regular (datapath) structures and also adding LUT functionality so that we can apply our circuits to logic synthesis as well as physical-design problems. Another important aspect to future work is to better capture the concept of locality in large circuits and to generate system-level circuits from combinational and sequential components.

Acknowledgements. Thanks to Stephen North and AT&T Bell Labs for academic license to use DOT[4] and Vaughn Betz for the use of his place-and-route software [1].

References

[1] V. Betz, *On biased and non-uniform global routing architectures and CAD tools for FPGAs.* Tech. Report in preparation. University of Toronto, 1996.

[2] J. Cong and Y. Ding, *FlowMap: An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-Table Based FPGA Designs*, IEEE Trans. CAD, 13 (June, 1994), pp. 1–12.

[3] 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, Feb., 1996, pp. 66–72.

[4] E. R. Gasner, E. Koutsofios, S. C. North, and K.-P. Vo, *A Technique for Drawing Directed Graphs*, IEEE. Trans. Soft. Eng., 19 (1993), pp. 214–230.

[5] M. D. Hutton, *Characterization and Automatic Generation of Digital Circuits and Systems.* Ph.D. Thesis in preparation, University of Toronto, 1996.

[6] M. D. Hutton and J. S. Rose, *Automatic Generation of Hierarchical Digital Circuit Systems.* Tech. Report in preparation. University of Toronto, 1996.

[7] B. S. Landman and R. L. Russo, *On a Pin Versus Block Relationship for Partitions of Logic Graphs*, IEEE Trans. Comp., C-20 (1971), pp. 1469–1479.

[8] Programmable Electronics Performance Corporation, *PREP PLD Benchmark Suite#1, V1.2.* 504 Nino Ave. Los Gatos, CA 95032, 1993.

[9] E. M. Sentovich *et.al*, *SIS: A System for Sequential Circuit Analysis.* Tech. Report No. UCB/ERL M92/41. University of California, Berkeley, 1992.

[10] 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.