Chapter 3

Characterization of Combinational Circuits

This chapter describes the statistical and structural characteristics that we have identified for combinational circuits.

Parts of this work are directly motivated by the generation problem. In order to generate benchmark circuits, we will need a *default* parameterization file, so we want to develop a statistical profile for relationships between parameters. For example, if the user asks simply for a circuit with 1000 nodes, we will need to choose a reasonable number of primary inputs and outputs, and a reasonable value for combinational delay. The complete set of default equations is in the file "comb.gen" shown in Appendix A of this dissertation.

Characterizations which describe the combinatorial structure of circuits, however, are of interest in their own right, and we propose a number of them here. Combinational shape, reconvergence and locality are all structural characterizations that are introduced in this thesis, and deal with the inherent structure in circuits which separates them from arbitrary graphs. In addition to becoming data for the circuit profile, the structural ideas will form the basis for the generation algorithm of Chapter 5.

For the empirical work here, we use the MCNC circuits. However, it is important to point out that the tool CIRC that we have produced to extract the characterization of a circuit is independent of the data; the user could use it on any collection of benchmark circuits, then redefine the default profile accordingly. CIRC is implemented to read circuit netlists in the Berkeley BLIF format [74], and output numerous statistical and structural characteristics.

As well, CIRC is able to do netlist translation, and output circuits in a number of other netlist formats (including Actel ADL [1], Altera AHDL/TDF [4] and Xilinx XNF [73]).

3.1 Empirical Data.

A large portion of the work in Chapters 3 and 4 is empirical, and for this we use the MCNC benchmark circuits. The use of the MCNC circuits is largely unavoidable, since they are the only large set of public benchmarks. We note that a user of the tools could profile their own (internal) circuits as the basis for an alternative defaults file. (See Appendix A.)

The MCNC benchmark circuits are a well-known set of combinational and sequential benchmarks available from http://www.cbl.ncsu.edu/. The circuits were converted from EDIF[1] to BLIF[2] using a modified conversion tool from MCNC. We did technology-independent optimization with SIS [62] (keeping the better result of script.rugged and script.algebraic) then technology mapped using FLOWMAP [13] into k-input lookup tables, for $k = 2..8$. 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 properties of circuits are sufficiently captured by the use of LUTs to determine valid characterizations without the added complexity of more technology-dependent libraries.

One issue that we do not fully explore in this work is the effect of this early optimization (CAD flow) on the exact statistical characterization which follows. For example, FLOWMAP is a delay based technology mapper, and it is not clear whether a different mapper would have changed some of our statistical results. Similarly, due simply to the volume of data, we spend most of our analysis on 4-LUT mapped MCNC circuits, largely because this is the most popular choice in the FPGA industry.

3.2 Basic Parameters of Combinational Circuits.

The characteristics in this first section are more for statistical purposes than to provide any new structural information about circuits.

[1] EDIF is a "standard" netlist format used in industry.
[2] BLIF is a format used by the Berkeley SIS tool, and commonly used in academia.

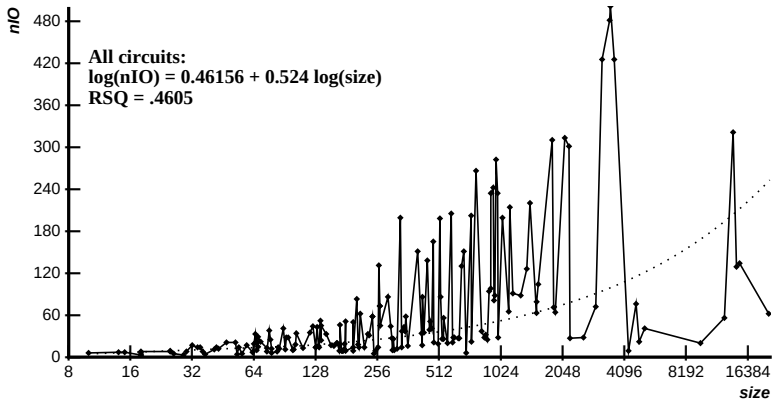


Figure 3.1: Size (2-LUTs) vs. I/O for MCNC circuits.

3.2.1 Circuit Size and I/O.

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 (n_{PI}) and outputs (n_{PO}). (Define $n_{\mathrm{IO}} = n_{\mathrm{PI}} + n_{\mathrm{PO}}$.) Using linear regression and experimentation, we have determined that a Rent-like functional relationship, $\log(n_{\mathrm{IO}}) = a + b \cdot \log(n)$ best captures the relationship between IOs and circuit size[3]. A simple linear relationship best describes the division of I/Os between inputs and outputs: $n_{\mathrm{PI}} = c + d \cdot n_{\mathrm{PO}}$. Figure 3.1 shows a plot of n vs. n_{IO}, and a least-squares regression line for the log-linear Rent relationship[4]. 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 truncated[5] Gaussian distributions around the best-fit line[6]. The actual equations are shown in the IOFrame section of comb.gen in Appendix A.

[3]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 Appendix A for the exact equations.

[4]Notice that the X-axis is shown with a log scale so that all points can be displayed with reasonable precision. Thus the visual variance around the regression line is deceptively large.

[5]Though the mean and variance can be determined exactly from the data, we shield ourselves from outliers by truncating the distribution before unrealistic values (in particular, negative values). It is also necessary for us to generate reasonably tame values, because a circuit which is an outlier in one parameter is often an outlier in all parameters, and choosing the parameterization independently cannot model this well.

[6]The regression line itself is not a strong predictor of the relationship between size and I/O, but this is not the point. Together with the Gaussian distribution of variance, we get a good probabilistic sample of a reasonable number of I/Os for a given size. Given the actual variance in the data, this is all that can be expected.

3.2.2 Nodes and Edges.

Two other dependent parameters of a circuit are the number of edges and the average fanin of the circuit. Looking at the data for 4-LUT mapped circuits, we see that average fanin varies from 2 to nearly 4, with a close to (truncated) Gaussian distribution centered around 3, and this is how we model it in the default profile. It is well known from technology-mapping literature that a circuit mapped to k-LUTs will not use all the inputs in each LUT unless $k = 2$, so this is to be expected.

As a byproduct of our experiments, we have observed that the final wirelength of a circuit after placement and global routing is much more highly correlated to the number of edges (equivalently average fanin) in the circuit than it is to the number of nodes. Though this might be easy to believe, it is quite interesting that utilization results for FPGAs are almost always specified in terms of the typical gate size of circuits which fit completely independent of the number of wires in the circuit. This suggests that a more accurate metric of "typical utilization" in an FPGA might be the *wire utilization* used, rather than the *logic utilization*, meaning that n_{edges} is probably a more indicative measure of circuit size than the number of nodes n.

3.2.3 Fanout Distribution.

Recall fanout(x) is the number of edges leaving a node x. A circuit's *maximum fanout* and *fanout distribution* (the number of nodes with fanout 0, 1, 2, etc.) is an important structural parameter which cannot be modeled by known methods in the theory of random directed acyclic graphs. Note that the *fanin* distribution is less interesting for technology-mapped circuits because they have an *a priori* constraint on fanin.

The maximum fanout and the fanout distribution for a selection of MCNC circuits is shown in Table 3.1. The first component gives the number of fanout zero nodes, which is less than or equal to the number of primary outputs (a primary output is not necessarily of fanout zero). A large proportion of the remaining nodes are fanout 1, with decreasing incidence as the fanout value gets higher. Most circuits with a reasonable number of nodes have some higher fanout values. Since these circuits are combinational, we do not have high-fanout clock, clear or reset signals to deal with, but even when discussing sequential circuits later we will ignore these special signals.

Using data from the entire benchmark set, we have developed a simple heuristic algorithm to generate reasonable fanout distributions given the circuit size, number of edges, max_fanout and number of I/Os. Essentially, we choose the n individual fanout values probabilistically from a discretized exponential distribution which is modified online to ensure that $\sum_i i \cdot fanout[i] = n_{edges}$ at completion.

Name	Size	Max_out	Fanout Distribution
cht	102	46	36 32 28 1 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 1 ...
9symml	106	34	1 94 2 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 2 ...
C1355	115	16	32 24 8 32 8 0 0 0 1 8 0 0 0 0 0 0 2
bw	137	66	25 72 17 9 1 4 1 2 0 0 0 0 0 0 0 1 0 0 0 0 ...
C1908	178	25	25 51 31 33 7 11 5 2 3 2 1 1 0 1 0 0 1 1 2 0 ...
C3540	481	66	21 235 88 37 11 21 15 3 9 5 1 1 2 0 1 1 14 2 3 4 ...
x3	512	122	99 250 80 29 12 3 7 2 6 3 3 0 0 0 1 1 3 1 1 1 ...
ex4p	514	26	14 360 27 16 15 11 22 2 5 2 5 5 4 2 5 4 0 1 0 1 ...
C6288	559	43	32 35 450 8 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ...
alu4	1536	249	8 1267 67 41 32 33 14 13 11 3 2 9 9 5 4 0 0 1 1 0 ...

Table 3.1: Fanout distribution for selected MCNC circuits.

Though we take a relatively simplistic approach to modeling the fanout distribution, we note that this type of distribution is nothing like what is seen for random graphs. For random directed acyclic graphs of the same size (nodes and edges) as the MCNC circuits **cht** and **ex4p**, we see fanout distributions of (23 19 18 23 19) and (79 67 75 66 83 77 67) respectively, which are nearly uniform. We point out that this is largely by construction, since natural models for such random directed graphs result in bounded fanin + fanout in order for the graphs to both be connected and to have a linear number of nodes. However, there are no known ways of generating random directed graphs having exponentially distributed fanout vectors which are connected and have a reasonable number of edges.

The heuristic algorithm mentioned above is the model for fanout distribution that we use in the default profile.

To some extent, the average fanout and the distribution of fanout values is dependent on the LUT size k used in technology mapping. A circuit mapped to 2-LUTs will have a much lower average fanout than a circuit mapped into 7-LUTs, in general: though more logic is stored in a LUT (reducing the overall number of edges), the computed value is then used by more other LUTs in the netlist, increasing the fanout value. As a basic rule, the

average fanout follows the average fanin, with variations occurring based on the distribution of I/Os and flip flops.

CIRC outputs a number of other degree-related statistics about a circuit, such as the average fanin and fanout for each combinational delay level, and the average fanout for primary inputs (and later flip-flops) as opposed to internal nodes. These are not used in the default profile, but we note that the information they provide is useful in the debugging of CAD tools, and in analyzing place and route anomalies occurring when the tool encounters outliers in the input.

3.3 Delay-Based Parameters of Combinational Circuits.

For a combinational circuit, 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, $d(G)$ (or just d), of a circuit is the maximum delay over all nodes in G. Using a similar empirical analysis to that previously mentioned, we have determined a stochastic relationship between delay d and circuit size n in which d is roughly $\log n$ on average.

Figure 3.2 shows a plot of size vs. combinational delay for 83 combinational MCNC circuits. The dashed function is the line $d = \log(n)$, representing the expected delay for a circuit with n nodes. The lower dotted line is $d = \log(\log(n))$, and the upper dotted line is $d = 3 \cdot \log(n) + \log(\log(n))$. Together these represent the lower and upper bounds on delay as modelled in the circuit profile[7].

3.3.1 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 distribution*, shape(G), of a circuit as the number of nodes at each combinational delay level. Figure 3.3 shows a small example circuit (**cm151a**), and its shape distribution (12, 4, 2, 2) displayed as a

[7] The dashed line is a best-fit regression line for the expected delay, and the default is to choose from a Gaussian distribution centered on this line. The two dotted lines represent the imposed truncation on the Gaussian distribution, i.e. the imposed upper and lower bound on the values which will be chosen. The imposed lower bound is $\log(\log(n))$ and the imposed upper bound is $n/3$. These upper and lower bounds given above and shown pictorially in the graph are chosen to include a majority of the points which are feasible, while excluding outliers (such as negative delay) which might otherwise occur. Note that modeling in this way underestimates the number of outliers often seen in practice, as evidenced in Figure 3.2.

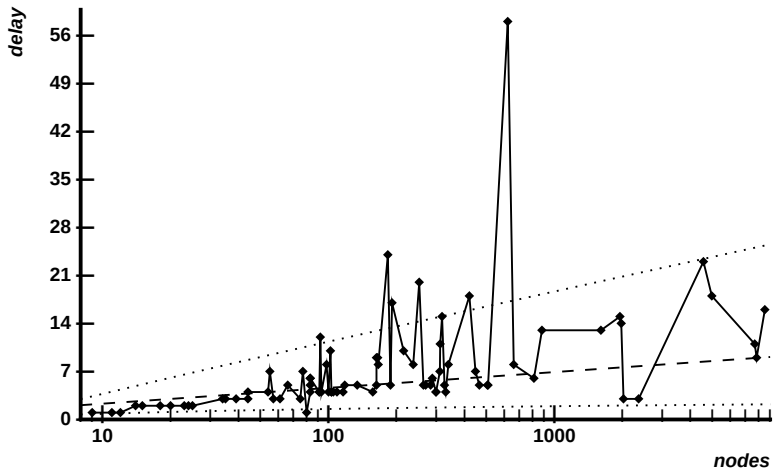


Figure 3.2: Size vs. combinational delay for MCNC circuits.

histogram. Note that even though the primary outputs are shown in circuit drawings we do not count them in determining delay or the shape distribution. Rather, we define "primary output" as a property on the fanin node. While these examples are mapped to 4-LUTs, the basic form of the distribution changes only proportionately for different LUT-sizes.

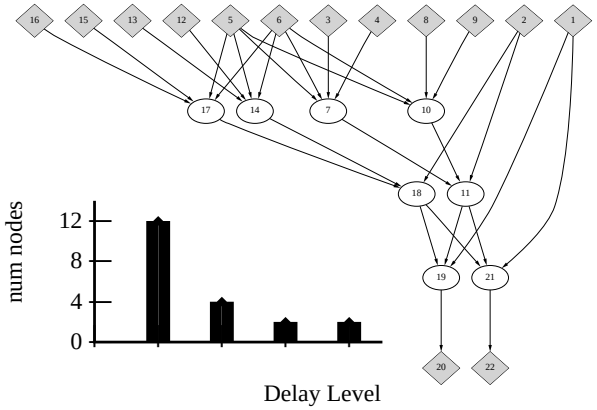
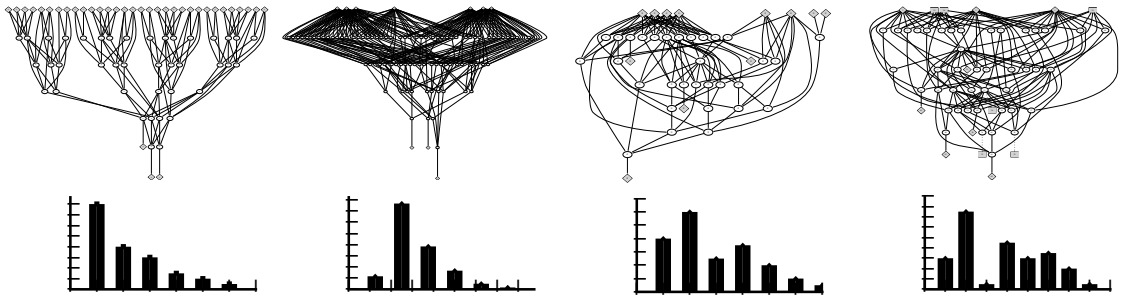


Figure 3.3: Shape distribution.

A characterization such as shape is not an obvious one to a circuit designer, who typically thinks of a design in terms of block diagrams, physical layout, or a set of boolean equations. However, looking at circuits from a graph-theoretic point of view, it is natural to try to draw the circuit in the plane with nodes divided into delay levels, and the importance of shape becomes clear.

The interesting thing about shape is that most circuits tend to have similar shapes. Random directed acyclic graphs from natural distributions tend, as a group, to have a different typical shape. Table 3.2 shows a sample of shape distributions for MCNC circuits,



comp	rd73	sqrt8ml	dk14 (L0)
(36)	(53)	(12)	(8)
"decreasing"	"conical histogram"	"two maxima"	"remaining cases"

Figure 3.4: Different shape distributions.

along with a qualitative classification of different shape functions. Figure 3.4 shows four shape classes and an example of each. Of the 109 combinational multilevel circuits in the MCNC set, 36 have a shape which is strictly decreasing from the primary inputs (as **comp**), 53 have a *conical* shaped histogram, fanning out from the inputs to an extreme point, then strictly decreasing (as **rd73**), 12 have the conical shape with a "bump" (as **sqrt8ml**) and only 8 did not fit into these categories. This distribution of shapes is fundamentally different from degree-constrained random graphs (discussed earlier in Section 2.2.3 and in Chapter 6) which tend, as a group, to almost always have a basically "flat" shape.

We performed experiments to determine whether there is any relationship between shape and routability metrics such as wirelength per edge. However, no significant correlation was found to exist for the MCNC data.

Name	Size	Delay	Shape Distribution
cht	102	2	47 44 11
9symml	106	6	9 57 24 7 6 2 1
C1355	115	4	41 24 8 10 32
bw	137	4	5 57 46 17 12
C1908	178	10	33 23 13 14 22 27 20 6 10 8 2
C3540	481	12	50 82 104 76 44 29 24 22 17 16 10 5 2
x3	512	5	135 202 123 40 10 2
ex4p	514	5	84 245 124 42 14 5
C6288	559	28	32 76 30 30 30 30 30 30 30 30 30 30 30 30 30 29 7 2 2 2 2 2 2 2 2 2 2 3 2
alu4	1536	7	14 692 518 198 80 21 11 2

Table 3.2: Shape distribution for selected MCNC circuits.

Though the example of Figure 3.3 shows both primary inputs at the last combinational

delay level and having zero fanout, neither is typical. We also extract and use the shape distribution of primary outputs (POShape) in the default profile of circuits. POShape is a vector of the number of output nodes at each combinational delay level.

3.3.2 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(G), and we define a related *edge length distribution.*

In the example of Figure 3.3 there are 24 edges of length 1, and 2 each of length 2 and 3, so the edge length distribution is (0,24,2,2,0). (Note the placeholder for absent length-0 edges; this is just so that we can have all vectors indexed similarly from 0).

Table 3.3 shows a sample of edge-lengths from the MCNC circuits. We find that almost all circuits have an edge-length distribution with a very similar structure: a large number of edges of length 1, and a quickly falling distribution over the combinational delay of the circuit. This type of distribution is not at all what one would expect of a random graph where the probability of any two pairs of edges being connected is the same. Empirically, such an edge length distribution is not common for random directed graphs arising from natural models (see Section 6.1.).

In the default profile, we model the edge length distribution by probabilistically sampling a discretized exponential distribution, which closely approximates this behaviour[8]

Name	Edges	Delay	Edge-Length Distribution
cht	102	2	0 202 0
9symml	106	6	0 271 41 6 6 0 0
C1355	115	4	0 216 32 0 32
bw	137	4	0 349 93 11 6
C1908	178	10	0 319 78 37 14 11 11 8 16 15 0
C3540	481	12	0 1017 317 143 28 18 13 14 13 6 0 5 1
x3	512	5	0 1071 139 49 8 2
ex4p	514	5	0 1248 167 8 3 0
C6288	559	28	0 1094 70 66 66 66 66 66 66 66 66 66 68 70 65 62 63 2 0 0 0 0 0 0 0 0 0 0 1 0
alu4	1536	7	0 4494 757 125 23 1 0 0

Table 3.3: Edge-length distribution for selected MCNC circuits.

[8]There are no appropriate statistical techniques to formalize this, so "closely approximates" means that the distributions appear reasonable when compared by hand.

3.3.3 Fanout Shape.

Another natural shape characterization is the distribution of *total* fanout by combinational delay level. The fanout distribution by delay level for our sample circuits is shown in Table 3.4. It is interesting that fanout by delay level is close to a strictly decreasing function for all of the circuits sampled. (However, note the exceptions in **C1908** and **C6288**.) Since the shape is conical for many of these circuits, we make the observation that the average fanout of primary inputs is higher than for other nodes. In fact, this is largely true when the number of nodes on a level is smaller than that of its succeeding level in the shape function.

Even though this distribution provides interesting information about the structure of combinational circuits, in practice it results in over specification in the profile. This is because the shape, edge-length and fanout distributions already mentioned constrain the delay-fanout enough that we can calculate tight bounds algorithmically. Thus, we do not currently generate delay-fanout as part of the statistical profile of combinational circuits.

Name	Edges	Delay	Delay-Fanout Distribution
cht	102	2	157 1 0
9symml	106	6	226 60 24 7 6 1 0
C1355	115	4	112 32 72 64 0
bw	137	4	267 124 56 12 0
C1908	178	10	167 53 36 87 67 51 17 19 10 2 0
C3540	481	12	558 338 204 142 85 74 53 42 37 21 18 3 0
x3	512	5	874 297 82 14 2 0
ex4p	514	5	868 376 135 33 14 0
C6288	559	28	1056 119 58 58 58 58 58 58 58 58 58 58 58 58 62 61 6 2 2 2 2 2 2 2 2 2 3 2 0
alu4	1536	7	2867 1700 529 197 79 20 8 0

Table 3.4: Delay-fanout distribution for selected MCNC circuits.

3.4 Reconvergence in Combinational Circuits.

Reconvergence occurs when multiple fanouts from a single node x in the circuit branch back together at a later point y—we say the circuit is *reconvergent at* y. Many circuits exhibit reconvergent fanout, but in widely varied degree, 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: the subgraph induced by all nodes reachable on a directed path from x. Figure 3.5 shows out-cone(a). Edges which are not in the out-cone, but are incident with nodes which are, are shown as bold dashed lines.

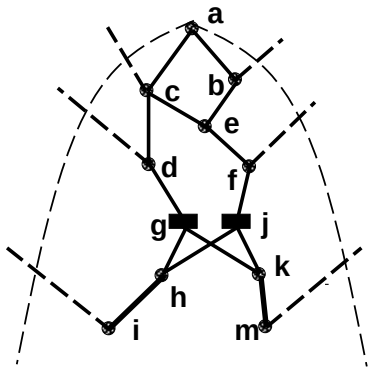


Figure 3.5: Reconvergence in combinational circuits.

For circuits mapped to 2-LUTs, 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):

$$R(x) = \frac{|\ \{y \in \text{outcone}(x) \text{ s.t. } y \text{ has fanin 2 in outcone}(x)\}\ |}{|\text{outcone}(x)|} \tag{3.1}$$

This value arises from its combinatorial interpretation. By Kirchoff's theorem [31, pp. 49-54], the numerator counts the $\log_2 t$ where t is the number of *spanning out-trees*[9] rooted at x in the directed graph representation of the circuit. Essentially, each reconvergent node represents a choice of two alternatives in the construction of a spanning out-tree, which multiplies the number of trees by two (adds 1 to $\log_2(t)$). Each non-reconvergent node represents a "required" in-edge, hence does not affect the number. The purpose of taking the logarithm is simply to obtain tractable numbers when dealing with large graphs. The denominator then scales that value with the size of the out-cone so that different graphs can be compared based on their relative amount of reconvergence, which otherwise would be dominated by the size of the circuit[10].

The intuitive argument for counting spanning out-trees is clear: a single spanning out-tree has zero reconvergence, and the number of spanning out-trees scales with the number

[9] A spanning out-tree rooted at r is a spanning tree such that each node, except the designated root node, has exactly one fanin. Hence each node lies on a unique directed path from the root.

[10] Analysis shows that there is no significant statistical correlation between R and n, so this adjustment is sufficient.

of ways that reconvergent fanout occurs in the circuit. This is even more compelling when we generalize the reconvergence calculation to sequential circuits in the next chapter.

For circuits mapped to k-LUTs, $k > 2$, the reconvergence calculation generalizes, both algorithmically and combinatorially, if we set the numerator as the sum, over all nodes y in the out-cone of x, of $\log_2(\text{fanin}(y))$. Thus $0 \leq R(x) \leq \log_2(k)$.

$$R(x) = \frac{\sum_{y \in \text{outcone}(x)} \log_2(\text{fanin}(y))}{|\text{outcone}(x)|} \tag{3.2}$$

To identify the reconvergence $R(G)$ present in an entire *circuit* G, we compute the weighted (by out-cone size) average of $R(x)$ for all primary inputs x in G. Thus $0 \leq R(G) \leq \log_2(k)$ continues to hold for circuits. In this way, highly reconvergent small portions of a circuit will not unduly affect the overall quantification.

The observed reconvergence numbers for the 198 combinational and sequential 2-LUT-mapped MCNC circuits vary between 0.0 and 0.92, 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 that intuitively simple, tree-like, logical functions have low R (e.g. **parity**: $R = 0.00$, **decod**: $R = 0.00$, **mux**: $R = 0.15$), and more complex functions have higher R (e.g. **alu2**: $R = 0.52$, **sqrt8ml**: $R = 0.53$). Combinational logic and the combinational parts of sequential arithmetic logic fall mostly in the range 0.0 to 0.6, whereas the combinational parts of finite state machines are mostly in the range 0.5 to 0.85 (9 of the 10 most reconvergent circuits are finite state machines). Table 3.5 shows the reconvergence numbers for a sample of combinational MCNC circuits for which we have some functionality information. Note that this information is inherently biased, because most circuits have no listed description and were left out of the table. Thus we can make only the vague observations about relative complexity of the logic.

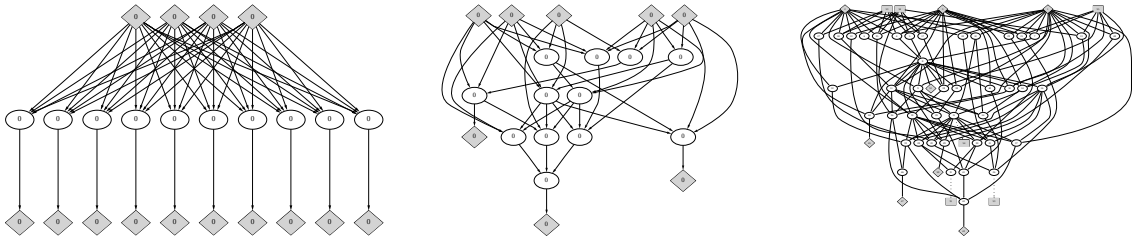
In a physical sense, there is a high degree of correlation between R and the other characteristics of a circuit; in particular, the number of edges (when $k > 2$), and the shape and out-degree functions. Using the examples of Figure 3.4, circuits which have an exaggerated conical shape, such as **rd73** ($R = 0.40$) and **sqrt8ml** ($R = 0.53$) tend to have higher reconvergence values, whereas circuits like **comp** ($R = 0.22$) 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. A conical shape arises because

Name	R	Description
parity	0.00	parity tree
decod	0.00	simple decoder
count	0.15	counter
mux	0.15	multiplexor
C1355	0.19	error correcting
my_adder	0.21	adder
C5315	0.26	ALU and selector
dsip	0.27	sequential encryption
des	0.30	data encryption
z4ml	0.30	2-bit adder
9symml	0.31	count ones in inputs
C2670	0.32	ALU and control logic
C7552	0.34	ALU and control logic
C880	0.36	ALU and control logic
s208	0.38	sequential multiplier
s838	0.41	sequential multiplier
s1196	0.41	sequential "logic"
C1908	0.44	error correcting
i10	0.47	combinational "logic"
sbc	0.47	sequential snooping bus controller
C3540	0.50	ALU and control logic
alu2	0.52	ALU
sqrt8ml	0.53	square root function
mult16a	0.54	sequential 16 bit multiplier
mult32b	0.54	sequential 32 bit multiplier
C432	0.58	priority controller
C6288	0.63	16 bit multiplier
apex4	0.63	combinational logic from a PLA
s400	0.63	sequential FSM: traffic light controller
clma	0.63	sequential bus interface
bbtas	0.76	finite state machine
pdc	0.79	finite state machine
s1488	0.83	finite state machine (controller)
dk16	0.89	finite state machine

Table 3.5: Reconvergence for selected MCNC circuits.

of a low I/O to logic ratio, natural because I/Os are "reused" over time in a sequential circuit.

Figure 3.6 shows examples of three different small circuits. The first, **cm42a** is a decoder, and has no reconvergence at all. The second, **rd53**, is combinational control logic, and has a reconvergence number of 0.40. The third is the first level of a finite state machine (we just converted flip-flops to primary inputs and primary outputs and dropped any logic past the flip flops). Its computation of reading the inputs and producing an encoded state has a reconvergence number of 0.69, the largest of the three.



cm42a R=0.00 | rd53 R=0.40 | dk14 (L0) R=0.69

Figure 3.6: Circuits with Varying Reconvergence.

3.5 Locality in Combinational Circuits.

To this point we have concentrated on *delay* as the fundamental characteristic of a circuit. Both the shape and edge length functions are delay based. This differs from previous work on wireability analysis, outlined in Section 2.2.2, which uses Rent's Rule and other stochastic measures of wirelength to describe the physical characteristics of a circuit.

In the generation process, it is clearly necessary to introduce some form of local clustering into a synthetic circuit. In this section we visit the issue of local structure in combinational circuits, with the goal of better understanding wirelength issues in the context of our existing delay based combinational model. Specifically, we will define metrics for wirelength and edge connections between delay levels and give an algorithm for ordering and positioning nodes within their combinational delay which allows us to calculate these metrics.

The best method of measuring the real wirelength and other routability parameters would likely be to execute placement and global routing on a gate array and measure the Manhattan wirelength, as would be performed by layout tools such as VPR [8], Altera's MAX+PLUS2 [4] or Xilinx PPR [73]. However, our purpose is to quickly determine a small amount of information necessary to characterize the locality in a circuit, not to do a complete and expensive physical layout.

Our process for extracting locality information is to determine an ordering of the nodes within each combinational delay level, and then an integer x-coordinate positioning for each node which respects the order: in other words, an embedding of the circuit graph on the integer grid, where the y-coordinate is constrained to be the node's combinational delay.

Given such a positioning $u.x$ for each node u, we can establish a number of metrics: Define *spread(i)* as the difference between the maximum and minimum x coordinates of

nodes on level i (i.e. the "width" of level i). Define *span(u)* for node u as the maximal distance between the coordinates of its fanins. Define *wirelength(u,v)*, for edge $e = (u, v)$ to be $|u.x - v.x|$, *wirelength(u)*, for node u, to be the sum over all fanins u of v of wirelength(u, v), and *wirelength(C)* for a combinational circuit C to be the sum, over all nodes u in C, of wirelength(u).

We note that the wirelength of a circuit in this sense is a layout into a shape structure. Thus it would be related to, but not necessarily the same value as, wirelength after embedding into a standard cell array, a gate array, or an FPGA. Empirically, though there is a strong linear relationship between the two forms of wirelength, the variance is large enough that the version based on shape would not, in itself, be a valid predictor of wirelength or routability in a gate array or FPGA.

To order and position the nodes for these wirelength and span calculations, we use an approach similar to that used by Gasner, North and Vo in the DOT package [30], used to draw many of the pictures in this thesis. The basic approach for ordering is to use the barycentric heuristic [22] to iteratively reduce crossing number between delay levels. We then diverge from the DOT approach to perform a more straightforward method of positioning nodes with integral coordinates which maintain the ordering but reduce wirelength. Sections 3.5.1 and 3.5.2 discuss these two aspects of the algorithm, then Section 3.5.3 discusses the results of executing the algorithm on combinational MCNC circuits.

3.5.1 Node Ordering Within Delay Levels

The problem of node ordering on a DAG G with delay d is to compute "good" orderings of the nodes at each level i, $0 \leq i \leq d$. The word "good" in the context of graph drawings is itself a new area of research, and there is no uniformly accepted metric of goodness. However, previous research [5, 22, 47] has determined that minimizing the *crossing number* not only yields drawings which are more viewable, but it also tends to illustrate symmetry and minimize the length of the drawn edges. Furthermore, since our ordering problem is similar to the placement problem of standard-cell layout, minimizing the crossing number is clearly desired. The crossing number of a graph and a given ordering is the number of pairwise crossing edges in the straight-line drawing of the graph when nodes are constrained in the y coordinate to their delay level and in the x coordinate to the determined ordering.

Figure 3.7 shows a drawing (by DOT [47]) of the MCNC circuit **comp** which illustrates

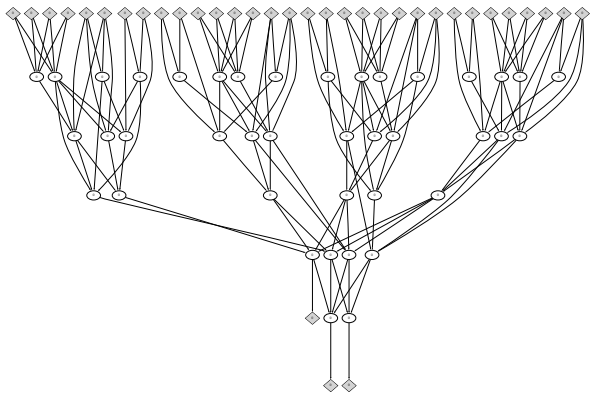


Figure 3.7: Minimizing crossings for a better "drawing."

the local effects of minimizing the crossing number. Though this drawing retains many crossings, a natural effect of the algorithm is to separate the loosely connected portions of logic. Our goal in this section is to take advantage of this separation in order to determine the amount of local structure present in a circuit. Because a drawing in this way corresponds to our delay-based model of circuits, this is also a natural way to impose local structure on a circuit later in the generation algorithm.

The problem of layout to minimize the crossing number is known to be NP-hard [21], even when $d = 1$ (i.e. the graph is bipartite and has two levels). Thus only heuristic algorithms are possible.

We will use a method similar to that originally used by Sugiyama *et. al.* [65], analyzed by Eades and Wormald [22] and used by Gasner *et. al.* [30] for the DOT program.

The basic algorithm is as shown in Figure 3.8. Note that "current_order" and "best_order" refer to data structures which hold the ordering for all levels of the graph.

On even passes, we treat level $i-1$ as fixed, and resort the nodes at level i based on the average ordinal value of their fanins. For odd passes we use level $i+1$ as fixed and look at fanouts. The initial order is simply a random ordering of nodes for each level.

The algorithm converges very quickly—about 10 iterations suffice for even large circuits (this is pointed out by Gasner *et. al.* [30] as well). The crossing number typically decreases by about a factor of 10 from the randomized to the "placed" version. We note that in random graphs generated as per Section 6.1, the crossing number decreases only by a factor of about 3.

```
best_order = a random order
compute crossing(best_order)
current_order = best_order
iter = 0
loop
    /* Compute a new current order */
    for (j = 0; j < d; j++)
        /* Working on combinational delay level j */
        if (iter is even)
            compute the average fanin index of each node at level j
            resort nodes at level j based on average fanin index
        else
            compute the average fanout index of each node at level j
            resort nodes at level j based on average fanout index
        end if
    end for
    iter++
    compute crossing(current_order)
    exit loop if crossing(current_order) > crossing(best_order)
    best_order = current_order
end loop
```

Figure 3.8: Algorithm to compute the crossing number.

Computing the crossing number.

In order to execute the heuristic algorithm above, we need to calculate the crossing number for edges between two combinational delay levels. The obvious approach is to examine each pair of edges to see if they cross, which can be accomplished in $O(n^2)$ time—we have $O(n)$ nodes and also $O(n)$ edges between any two delay levels, under the assumption of constant fanin (otherwise the obvious algorithm becomes $O(n^4)$). For large circuits, a quadratic algorithm is too expensive. Fortunately, we can give an easy to implement $O(n \log n)$ algorithm. To our knowledge, no such algorithm has been previously given for computing the crossing number.

Problem: Given a bipartite graph $G(X, Y)$ and sorted orders $1..|X|$ and $1..|Y|$ for the nodes of X and Y, determine the number of pairwise crossing edges, that is the number of pairs of edges (x_1, y_1) and (x_2, y_2) such that $x_1 < x_2$ and $y_2 < y_1$ in the respective orderings of X and Y.

Our solution uses a divide and conquer approach which, interestingly enough, actually allows us to count more crossings than we examine (i.e. we can count $O(n^2)$ crossings in $O(n \log n)$ time.

Let $n_x = |X|$ and $n_y = |Y|$, and let x_i (y_i) denote the i'th node in the sorted order of X (Y).

Define the following sets of nodes:

A: nodes x_i in X such that $i \leq \frac{n_x}{2}$

B: nodes x_i in X such that $i > \frac{n_x}{2}$

C: nodes y_i in Y such that $i \leq \frac{n_y}{2}$

D: nodes y_i in Y such that $i > \frac{n_y}{2}$

Then we can classify each edge as AC, AD, BC or BD. There are $4 \cdot 4 = 16$ types (combinations) of edge crossings.

We calculate the number of crossings from X $(A + B)$ to Y $(C + D)$ by dividing the edges into their categories (trivially in $O(n)$ time) and decomposing the problem as follows:

$$
\begin{aligned}
&\text{crossings}(A + B, C + D) = \\
&\qquad \text{crossings}(A + B, C) && \text{/* recursive call */} \\
&\qquad + \text{crossings}(A + B, D) && \text{/* recursive call */} \\
&\qquad + \text{num\_cross}(AC - AD) && \text{/* separate computation */} \\
&\qquad + \text{num\_cross}(BC - BD) && \text{/* separate computation */} \\
&\qquad + |AD| \cdot |BC| && \text{/* sizes only */}
\end{aligned}
$$

The recursive call "crossings$(A + B, C)$" refers to the sub-problem on the nodes (and induced edges) not incident on D. The call to num_cross$(AC - AD)$ will be a separate routine to count all crossings between an AC and an AD edge, and no others.

Since C and D partition Y evenly, each recursive call is on at most one half of the maximum edges to the preceding call. Thus, as long as we can find a linear time algorithm for num_cross() the entire algorithm will be $O(n \log n)$.

The cases for num_cross() are symmetric, so we will work on num_cross$(AC - AD)$ only. Assume that the edges have been divided into AC and AD edges already (easily $O(n)$ time), and are still sorted by x_i value in the ordering. Then an AC edge (x_i,y_j) and AD edge (x_k,y_l) cross if and only if $i > k$. We know that $j < l$ from the edge classes.

We take a single pass through A, and count the number of AC edges at each location i. Then we scan again, summing, to calculate the number of AC edges *to the right of* location

i. In a third pass, we look at every AD edge, which necessarily crosses the number of AC edges with x coordinate to the right of the current location, which is the previous sum vector. This correctly calculates num_cross($AC - AD$) in linear time. Note that we do *not* count the $AC - AC$ edges here, or we would be double counting them.

The proof that the algorithm works is a simple case analysis.

AC-AC	–	counted only in crossings($A + B, C$)
AC-AD = AD-AC	–	counted only in num_cross($AC - AD$)
AC-BD = BD-AC	–	cannot cross
AC-BC = BC-AC	–	counted only in crossings($A + B, C$)
AD-AD	–	counted only in crossings($A + B, D$)
AD-BD = BD-AD	–	counted only in crossings($A + B, D$)
AD-BC = BC-AD	–	must cross; counted in the product
BC-BD = BD-BC	–	counted only in num_cross($BC - BD$)
BC-BC	–	counted only in crossings($A + B, C$)
BD-BD	–	counted only in crossings($A + B, D$)

We conclude that crossings($A + B$,$C + D$) can be calculated in $O(n \log n)$ time.

3.5.2 Coordinate Positioning of Nodes

To position nodes, we perform another iterative step.

From the previous step, the order of nodes within each delay level is fixed. Define *width* to be equal to the maximum size of any delay level, and coordinates $u.x$ for every node u, which equally proportions the nodes at level i across *width*.

The iterative step is similar to the ordering algorithm, except that we do not exchange nodes, just move them closer together or further apart within the ordering. On even iterations we define the *centre* of a node u as the average x coordinates of its fanins. On odd iterations we use its fanouts.

For each node u at level i, we compute centre(u), and move $u.x$ as far as possible to center(u), without going past u's neighbour.

Wirelength, as previously defined, is the sum of the lengths of each edge. The length of an individual edge is the difference in the x coordinates of its endpoints.

As in the ordering step, it takes only a small number of iterations for the wirelength to

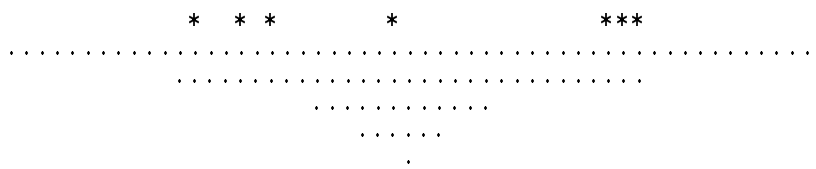


Figure 3.9: Locality placement for **rd73**.

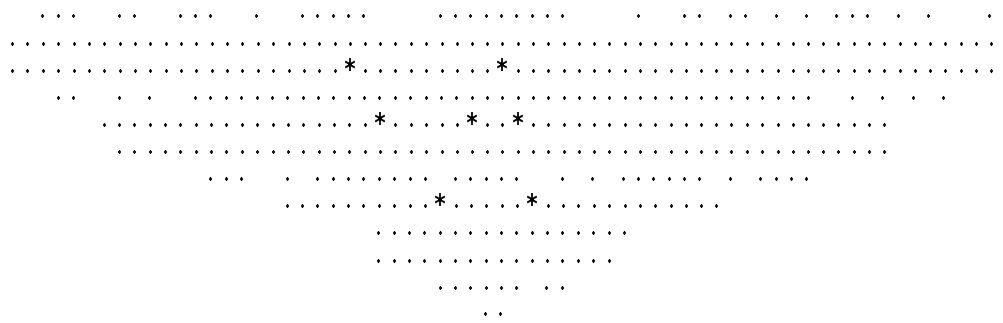


Figure 3.10: Locality placement for **C432**.

hit a minimum. At that point, we can also calculate the other metrics (span and spread) mentioned earlier.

3.5.3 Discussion

Though our goal in doing this pseudo placement is to extract metrics like spread and average span, it is interesting to see the effects of the placement on real circuits. We note that a complete algorithm that takes into account room for edges to be drawn would result in node coordinates that mimic the results of DOT.

Figure 3.9 shows a drawing of the nodes in **rd73**. A '.' indicates a node position, and a '*' indicates a node which has fanout greater than ten. We see a very balanced local structure below the inputs level, but a wide spreading of the primary inputs.

Figure 3.10, on the other hand, shows a circuit which has a slightly less balanced structure. In addition to high-fanout nodes, we see "holes" in the layout which indicate areas where nodes are drawn apart from their neighbours by local structure.

Figure 3.11 shows a structure which is further from balanced. We observe the wide spread of nodes at delay 1, for example.

Our final example is shown in Figure 3.12. This is a circuit which exhibits a great deal of local structure. We observe the tree-like way that terms are collected from the inputs to

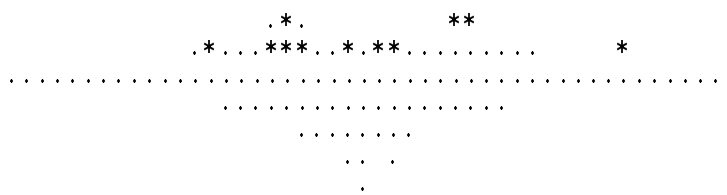


Figure 3.11: Locality placement for **rd84**.

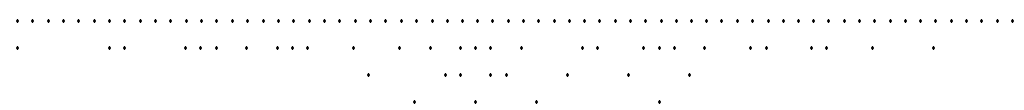


Figure 3.12: Locality placement for **i3**.

outputs: a wide spread, with lots of holes, as the delay increases. This indicates that nodes are more closely related to their close neighbours in index value.

The metrics of average in span and spread for each delay level values can be seen as quantifications of the locality present in the circuits: A high average node span indicates that nodes draw from a wider range of fanins, and that the circuit is less local than would otherwise be the case. The spread of a level, compared to the number of nodes it has, gives information about how closely the nodes at a level share fanins and are pushed together by the wirelength minimization process.

An important aspect of locality that these particular metrics (span and spread) do *not* capture is edge crossings, in particular the distribution of crossings over x coordinate "slices" of the drawing. It would be very useful in the generation algorithm to have more information of this type, but we leave this particular topic to future work. As well, though we can extract this information from specific existing circuits, we have not yet investigated methods to model this type of locality in the default profile (though this could be done). Thus it is currently useful only for generating "clone" circuits, as will be discussed in later chapters.

It is important that the locality algorithm is fast. Extraction of local information from a medium sized circuit such as **alu4** uses one tenth the cpu time of a complete place and route[11].

[11] This does not necessarily make the method a competitor for standard placement algorithms, because we are not restricting the placement to a minimal size square grid.