Balancing Interconnect and Computation in a Reconfigurable Computing Array (or, why you don't really want 100% LUT utilization)

André DeHon
Berkeley Reconfigurable, Architectures, Software, and Systems
Computer Science Division
University of California at Berkeley
Berkeley, CA 94720-1776
<andre@acm.org>

Abstract

FPGA users often view the ability of an FPGA to route designs with high LUT (gate) utilization as a feature, leading them to demand high gate utilization from vendors. We present initial evidence from a hierarchical array design showing that high LUT utilization is **not** directly correlated with efficient silicon usage. Rather, since interconnect resources consume most of the area on these devices (often 80-90%), we can achieve more area efficient designs by allowing some LUTs to go unused—allowing us to use the dominant resource, interconnect, more efficiently. This extends the "Sea-of-gates" philosophy, familiar to mask programmable gate arrays, to FPGAs. Also introduced in this work is an algorithm for "depopulating" the gates in a hierarchical network to match the limited wiring resources.

1 Introduction

The ability of an FPGA to support designs with high LUT usage is regularly touted as a feature. However, high routability across a variety of designs comes at a large expense in interconnect costs. Since interconnect is the dominant area component in FPGA designs, simply adding interconnect to achieve high LUT utilization is not always area efficient. In this paper, we ask:

- *Is an FPGA with higher LUT usage more area efficient than one with lower LUT utilization?*
- That is: *Is LUT usability directly correlated with area efficiency?*

Our results to date suggest that this is often not the case—achieving high LUT utilization can often come at the expense of greater area than alternatives with lower LUT utilization. While additional interconnect allows us to use LUTs more heavily, it often causes us to use the interconnect itself less efficiently.

To answer this question, we proceed as follows:

1. Define an interconnect model which allows us to vary the richness of the interconnect.
2. Define a series of area models on top of the interconnect model to estimate design areas.
3. Develop an algorithm for mapping to the limited wiring resources in a particular instance of the interconnect model.
4. Map circuits to a range of points in the interconnect space, and assess their total area and utilization.
5. Examine relationship between LUT utilization and area.

To appear in the Seventh International Symposium on Field-Programmable Gate Arrays, February 21–23, Monterey, CA.

2 Relation to Prior Work

Most traditional FPGA interconnect assessments have been limited to detailed population effects [1] [15]. In particular, they let the absolute amount of interconnect (*i.e.* number of wiring channels or switches) float while assessing how closely a given population scheme allows detailed routing to approach the limit implied by global routing. They also assume that the target is to fully populate the LUTs in a region of the interconnect.

Instead, we take the viewpoint that a given FPGA family will have to have a fixed interconnect scheme and we must assess the goodness of this scheme. To make maximum use of the fixed interconnect, in regions of higher interconnect requirements where the design is more richly connected than the FPGA, we may have to use the physical LUTs in the device sparsely resulting in a depopulated LUT placement. This represents a "Sea-of-Gates" usage philosophy as first explored for FPGAs in University of Washington's Triptych design [4].

For the sake of illustration, consider a design which has a small, but heavily interconnected controller taking up 20% of the LUTs in the design. The rest of the design is a more regular datapath which does not tax interconnect requirements. If we demanded full population, we would look at the interconnect resources necessary to fully pack the controller, and those requirements would set the requirements for the entire array. However, the datapath portion of the chip would not need all of this interconnect and consequently would end up with much unused interconnect. Alternately, we can spread out the controller, ignoring some LUTs in its region of placement, so that the whole FPGA can be built with less interconnect. Now, the controller may take up 30% of the device resources since it cannot use device LUTs 100% efficiently, but the whole device is smaller since it requires less interconnect.

Recently, NTT argued for more wires and less LUTs [17], and HP argued for rich interconnect which will meet or exceed the requirements of logic netlists [2]. Earlier Triptych showed density advantages over traditional alternatives with partially populated designs [4]. The NTT paper examined two points in the space, while HP and University of Washington each justified a single design point. In this paper, we build a model which allows us to explore the tradeoff space more broadly than a few isolated design points. The model is based on a hierarchical network design and captures the dominant switch and wire effects dictating wire area. This generalization, of course, comes at the cost of modeling the design space more abstractly than a particular, detailed FPGA design.

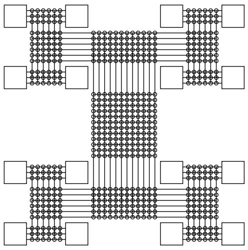


Figure 1: Tree of Meshes

We will be using a hierarchical interconnect scheme as the basis of our area model. Agarwal and Lewis's HFPGA [1] and Lai and Wang's hierarchical interconnect [11] are the most similar interconnect schemes proposed for FPGA interconnect. As noted above neither of these studies made an attempt to fix the wiring resources independent of the benchmark being studied as we are doing here. To permit a broad study of interconnect richness, our interconnect scheme is also defined in a more stylized manner as detailed in the next section.

3 Interconnect Model

The key requirements for our interconnect model is that it:
- represent interconnect richness in a parameterized way
- allows definition of a reasonable area model

To meet these goals, we start with a hierarchical model based on Leighton's Tree of Meshes [13] or Leiserson's Fat Trees [14]. That is, we build a tree like interconnect where the bandwidth grows toward the root of the tree (See Figure 1). We use two parameters to describe a given interconnect scheme:

1. c = the number of base channels at the leaves of the tree
2. p (α) = the growth rate of interconnect toward the root

Note that we realize p by using one of two kinds of stages in the tree of meshes:

- non-compressing (2:1) stages where the root wires are simply equal to the sum of root wires from the two children so there is no net bandwidth reduction
- compressing (1:1) stages where the root wires are the same as each of the root wires from the children, so that only half of the total children wires can be routed upward

By selecting a progression of these stages we can approach any bandwidth growth rate (See Figure 4).

If we use a repeating pattern of stage growths, we approximate a geometric bandwidth growth rate. That is, a subtree of size $2 \cdot n$ has 2^p times as much bandwidth at its root as a subtree of size n, or every tree level has $\alpha = 2^p$ more wires than its immediate children. This is roughly the model implied by Rent's Rule [12] ($IO = c \cdot N^p$). More precisely, it represents a bifurcator as defined by Bhatt and Leighton [3] (See Figure 2).

Intuitively, p represents the locality in interconnect requirements. If most connections are purely local and only a few connections come in from outside of a local region p will be small. If every gate in a region had a unique signal coming from outside the region, then $p \to 1.0$. So think of p as describing how rich our interconnect needs to be. If $p = 1$, we are effectively building a crossbar with no restrictions. If $p = 0$, we are building a 1D systolic array or pure binary tree whose IO bandwidth does not grows as the array grows.

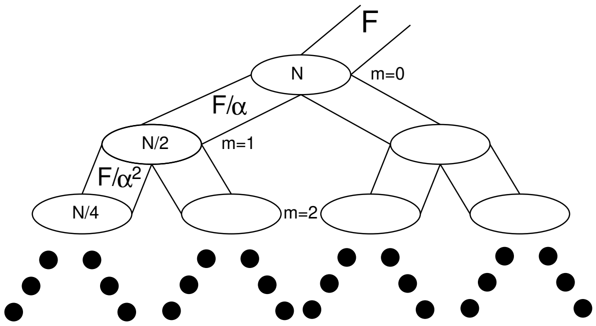


Figure 2: (F,α)-bifurcator

4 Area Effects

For our basic area model, we perform a straightforward layout of the elements shown in Figure 4. That is, we have:
- Logic Block of size A_{pe}
- Switches of size A_{sw}
- Wires of pitch WP

Each subtree is built hierarchically by composing the two children subtrees and the new root channel. Channel widths are determined by either the area required to hold the switches or the width implied by the wire channels, depending on which is greater. We assume a dedicated layer for each of horizontal and vertical interconnect. The result is the "cartoon" VLSI layout as shown in Figure 3.

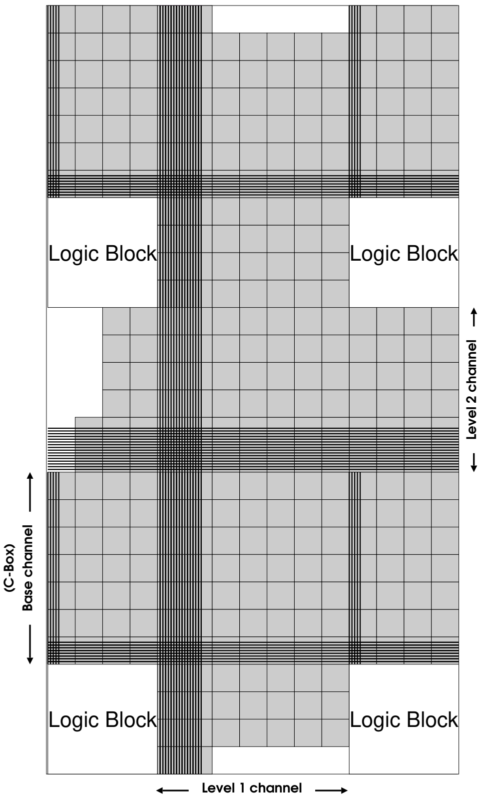
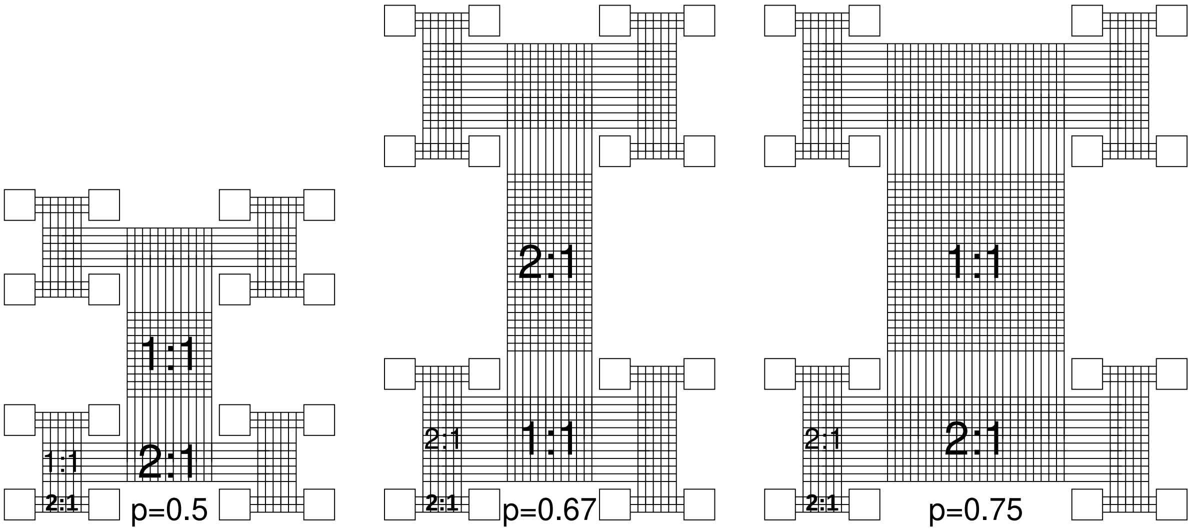


Figure 3: "Cartoon" Layout of Hierarchical Interconnect



Note that the number of base channels (c) is 3 in all these examples.

Figure 4: Programming Growth for Tree of Meshes

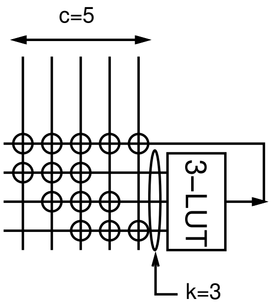


Figure 5: c choose k LUT Input Population ($c = 5$, $k = 3$)

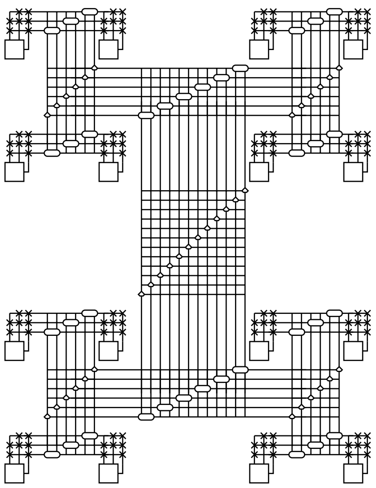


Figure 6: Linear Switchbox Population for Hierarchical Interconnect

Typical values for an SRAM programmable device:[1]

- $A_{pe} = 40\text{K}\lambda^2$ — this would hold 16 memory bits for a 4-LUT ($16\times1.2\text{K}\lambda^2$/SRAM-bit$\approx20\text{K}\lambda^2$) plus a the LUT multiplexor and optional output flip-flop ($13\text{K}\lambda^2$ in [5], $15\text{K}\lambda^2$ in [8]).
- $A_{sw} = 2.5\text{K}\lambda^2$ for a pass transistor switch (including its dedicated SRAM programming bit) — to model mask or antifuse programmable devices, we would use a much smaller size for this parameter.
- $WP = 8\lambda$ for the metal 2 or metal 3 wire trace and spacing

We assume the channels are populated with c choose k input selectors [7] on the input and have a fully populated output connection (See Figure 5). Switch boxes are either fully populated or linearly populated (see Figure 6) with switches.

Figure 7 shows cartoon layouts for 3 different choices of p, highlighting the area implied by each choice. Two things we can observe immediately from this simple model comparison:

- For reasonable parameters, interconnect requirements dominate logic block area; *e.g.* at $c = 6$, $p = 0.67$, a design with 1024 LUTS has only 5% of its area in LUTs (estimated area per LUT including interconnect is $\approx750\text{K}\lambda^2$) — while this is a simple area model, the area and ratio are not atypical of real FPGA devices; they are also consistent with prior studies (*e.g.* 6% for 600 4-LUT design in [5]).

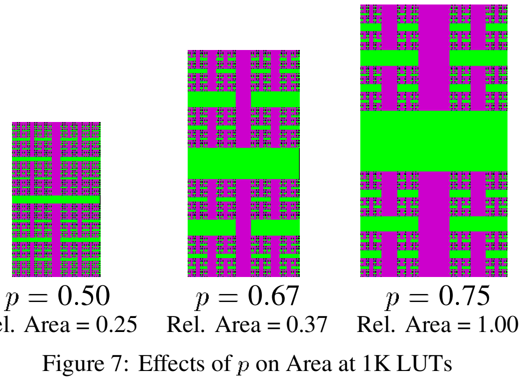


Figure 7: Effects of p on Area at 1K LUTs

[1] λ = half the minimum feature size for a VLSI process. Assuming linear scaling of all features, λ^2 area should be the same across processes.

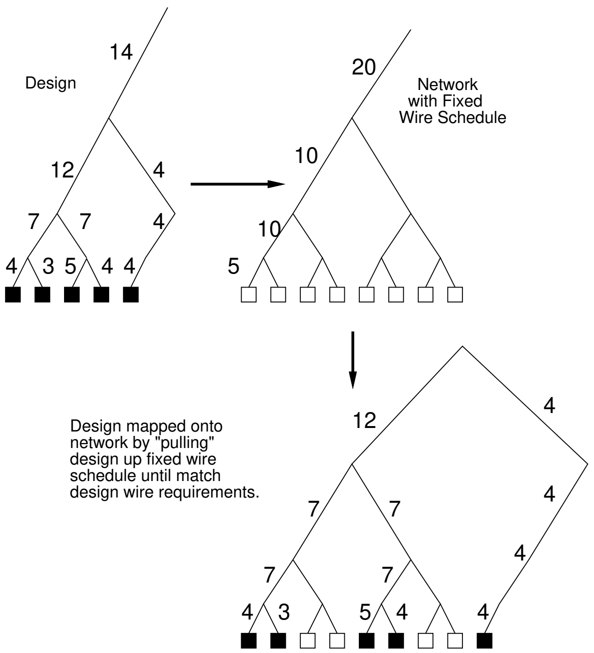


Figure 8: “Pulling” design up tree to match fixed wire schedule

- Interconnect parameter richness has a large effect on total area.

To further build intuition, let's assume for a moment that a design can be perfectly characterized by a growth exponent p. If the growth exponent for the interconnect matches the growth of the design ($p_{interconnect} = p_{design}$), then the network will require minimum area. What happens if these two are not perfectly matched? There are two cases:

- $p_{interconnect} > p_{design}$ — we have more interconnect than necessarily. The design can use all the LUTs in the network, but the network has more wires. As a result, the area per LUT is larger than the matched case—that is, mapping the design on the richer interconnect takes more area than the matched design case.

- $p_{interconnect} < p_{design}$ — we have less interconnect than necessary. We cannot pack the design into a minimum number of LUTs in order to fit the design. Instead we must pull the design up the tree, effectively depopulating the logic blocks, until the tree provides adequate connectivity for the design (See Figure 8). As a result, we have leaves in the tree which are not fully utilized. As we will see, this also takes more area than the matched design case.

Figure 10 shows the area overhead required to map various designs onto interconnects with various growth factors. As we expect, it shows that the matched interconnect point is the minimum point with no overhead. As we go to greater or lesser interconnect offered by the network, the area overhead grows, often dramatically.

5 Design Requirements

In practice, of course, c and p values are a rough characterization of the interconnect requirements for a real design. With multiple subgraphs of a given size (subtrees at the same height in the tree) we get more than one I/O to subgraph relationship. Further, the growth is seldom perfectly exponential. Finally, even asking if a graph has an (F, α)-bifurcator is an NP-hard problem. So, the bifurcations we construct are heuristic approximations biased by the tools we employ.

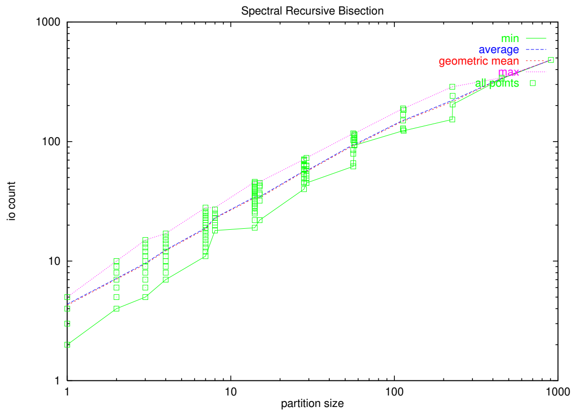


Figure 9: I/O versus Partition size graph for `i10`

Figure 9 shows the I/O versus subgraph relationship for the one of the IWLS93 benchmark (`i10`).

- Mapped for area with SIS [16] and Flowmap [6]
- Recursively bisected using a single Eigenvalue spectral partitioner

The recursive bisection approximates the natural bandwidth versus subtree sizes which exist in the design. We see the I/O to subgraph relationship is not 1:1. We also see that the max and average contours can be matched well to a geometric growth rate (*e.g.* Rent's Rule—average $c = 5$, $p = 0.7$; max $c = 7$, $p = 0.7$).

The left of Figure 11 shows the I/O versus subgraph relationship for all the IWLS93 benchmarks area mapped to 2000 or fewer LUTs using SIS, Flowmap, and spectral partitioning as above. On the right it shows the distribution of Rent parameter estimates for these benchmarks. Here we see that while we may be able to pick "typical" c and p values, there is a non-trivial spread in interconnect requirements across this set of designs.

6 Mapping to Fixed Wire Schedule

We have now seen that we can define a parameterized interconnect model with a fixed wire schedule. Designs have their own requirements which do not necessarily match the fixed wire schedule available from a device's interconnect. When the device offers more interconnect than a design needs, mapping is easy, we simply place the design on the interconnect and waste some wires. However, if the design has more interconnect needs than the device provides, how do we map the design to the device?

As suggested in Figure 8, we can start with the recursively bipartitioned design and simply pull the whole design up the tree until all the interconnect wires meet or exceed the design requirements. However, keeping the groupings originally implied by the recursive bisection is overly strict. In particular, re-associating the subgraphs based on interconnect availability can achieve tighter packings (See Figure 13). That is, we do not really want a bisection of the LUTs, but a bisection of the total capacity including both interconnect and LUTs. Intuitively, the size of a subgraph is determined by the greater of its LUT requirement and its interconnect requirement relative to the fixed wire schedule of the device.

To attack the problem of regrouping subtrees to fit into the fixed wire schedule, we introduce a dynamic programming algorithm which determines where to split a given subgraph based on the available wire schedule. That is, we start with a linear ordering of LUTs. Then, we ask where we should cut this linear order-

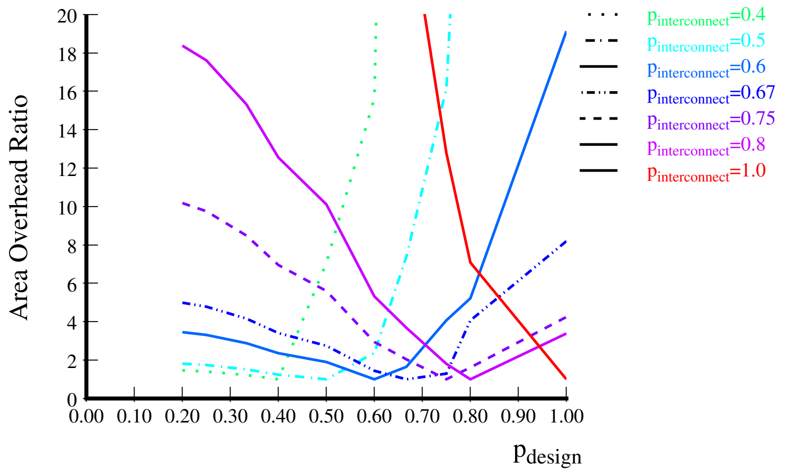


Figure 10: Theory: effects of mismatched between interconnect network p and design requirements

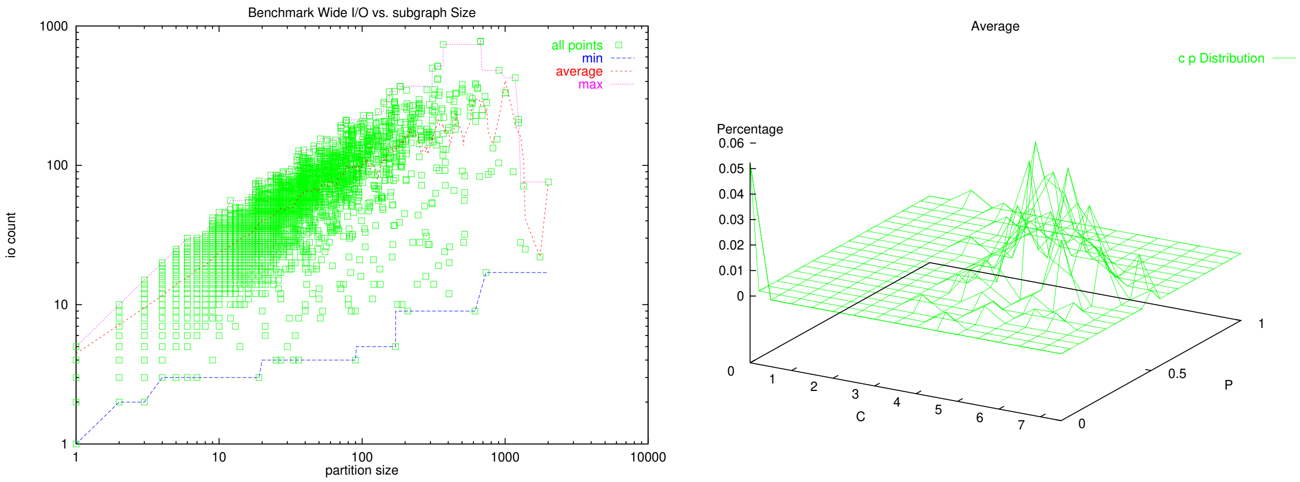


Figure 11: I/O versus Partition size graph for Benchmark Set

```
// size[start,finish] represents the smallest subtree which will
//          contain the set of LUTs between position start and finish
// uniqueio(o,i,j) returns the number of unique nets which appear both in the subrange i→j,
//          and outside of that range
o = order all LUTs
for i=0 to o.length
        size[i,i] ← size(1,unique(o,i,i)) // base case ≡ single LUT subtrees
for len=2 to o.length
        for start=0 to o.length-len // process all subranges of specified length
                minsize=MAX
                end=start+len-1
                isize=uniqueio(o,start,end)
                for mid=start+1 to end // search for best split point
                        msize=1+max(size[start,mid],size[mid+1,end])
                        size=max(msize,iolevel(isize))
                        minsize=min(size,minsize)
                size[start,end]←minsize
// final result is size[0,o.length-1]
```

Figure 12: Dynamic Programming Algorithm to Map to Fixed Wire Schedule

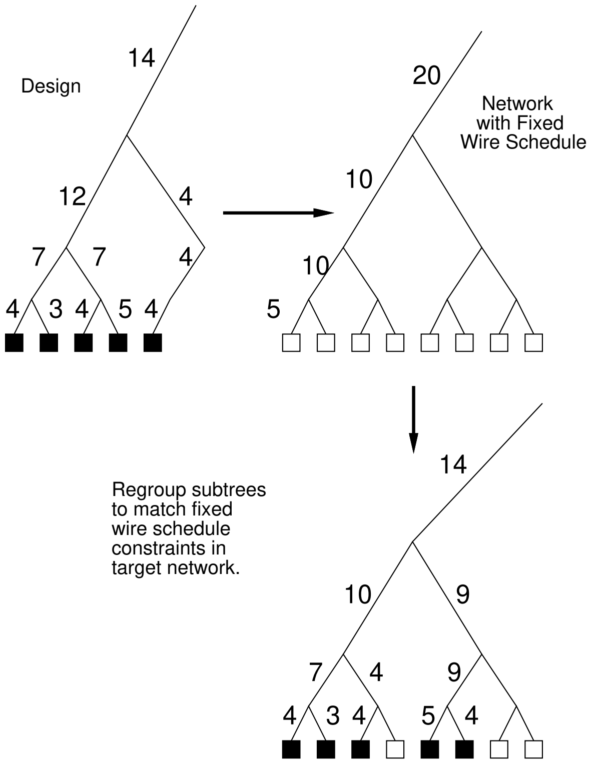


Figure 13: Re-associating Subgraph clusters to match Fixed Wire Schedule

ing of LUTs into two subtrees in order to minimize the total area required—typically, minimizing the heights of the two subtrees. Each of the subtrees are then split in a similar manner. To make the decision of where to cut a subtree, we examine all cut points. As long as we have a single linear ordering for LUTs, this is very similar to the optimal parenthesis matching problem. In a similar manner, we can solve this problem with a dynamic programming algorithm.

The dynamic programming algorithm (Figure 12) finds the optimal subtree decomposition **given** the initial LUT ordering. The trick here, and the source of non-optimality, is picking the order of the LUTs. For this we use the 1D spectral ordering based on the second smallest Eigenvalue which Hall shows is the optimal linear arrangement to minimize squared wire lengths [10].

Figure 14 shows why the single linear ordering is non-optimal. Here we see a LUT B placed to minimize its distances to A, C, and D. The order is such as to keep B, C, and D together for cut 3. However, if we take cut 4, then it would be more appropriate to place B next to A since we have already paid for the wires to C and D to exit the left subgroup. However, as long as we are using a single linear ordering, we do not get to make this movement after each cut is made. In general to take proper account of the existing cut, we should reorder each of the subgraphs ignoring ordering constraints originally imposed by the wires which have already been cut.

To avoid this effect, we would have to reorder each subtree after each cut is made. In addition to increasing the complexity of each cut, this would destroy the structure we exploited to apply dynamic programming—that is, the sub-problems would no longer be identical. Of course, since the spectral partitioning does not even give an optimal cut point for the bisection problem, the ordering effect alone is not the only element of non-optimality here.

There is certainly room for algorithmic improvement here. The

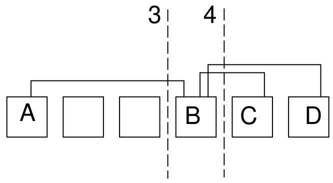


Figure 14: Example showing the limitations of a Single Linear Ordering

results are, nonetheless, good enough to give us interesting depopulations as we will see in the next section.

7 Results from Mapping

Putting it all together:

- Start with the area targeted SIS and Flowmap 4-LUT networks for the IWLS93 benchmarks under 2000 4-LUTs.
- Order using the second smallest Eigenvalue.
- Map to fixed schedule with the dynamic programming algorithm; The results are shown in terms of relative number of LUTs in the top left of Figure 15.
- Apply an area cost model such as shown in top right of Figure 15.
- Result is the relative area map shown at the bottom of Figure 15.

Figure 15 shows that there is a minimum area point across the benchmark set. For our linear switch population model, this occurs at $c = 6$, $p = 0.6$. As our theory predicts, too much interconnect and too little interconnect both account for area overheads over the minimum. Notice that the only points where the entire benchmark achieves full utilization are $c = 10$, $p \geq 0.75$ and $p = 0.8$, $c \geq 7$, all points which are above the minimum area point.

Table 1 examines the effects of picking a particular point in the c-p-design space. For each design in the benchmark set, we can compute the c,p-point which has minimum area. We can then look at the overhead area required between the "best" c, p, picked for the individual design, versus the best c, p for the entire benchmark under certain criteria. For the linear switch population case, we see that average overhead between the benchmark minimum and each benchmark's best area is only 23% and that corresponds to an average LUT utilization of 87%. Similarly, we see that picking the smallest point where we get 100% device utilization results in almost 200% area overhead. We see different absolute numbers, but similar trends with other area models.

Given the range of partition ratios and cut sizes we saw in Figure 11, it is not that surprising that the full utilization point is excessive for many designs and leads to many area inefficient implementations. Figure 16 shows a slice in p-space for the single design `i10` whose I/O versus subgraph size curve we showed in Figure 9. Notice that even for this single design, the minimum area point does not correspond to full utilization. In fact, the minimum area point is actually only 50% of the area of the full utilization point. So, even for a single design allowed to pick the network parameters c, p which minimizes device area, full LUT utilization does not always correspond to better area utilization. We see here that the effects of varying wire requirements, which we described in Section 2, do actually occur in designs.

In the previous section, we noted that the fixed wire schedule mapping algorithm in use is not optimal. It is worth considering how a "better" algorithm would affect the results presented here. A "better" algorithm could achieve better LUT utilization for the points where depopulation occurs. For the points on the graph where no depopulation occurs, a better algorithm could offer no improvement. As a result, we expect a better algorithm to magnify these effects—making the depopulated designs tighter and take less

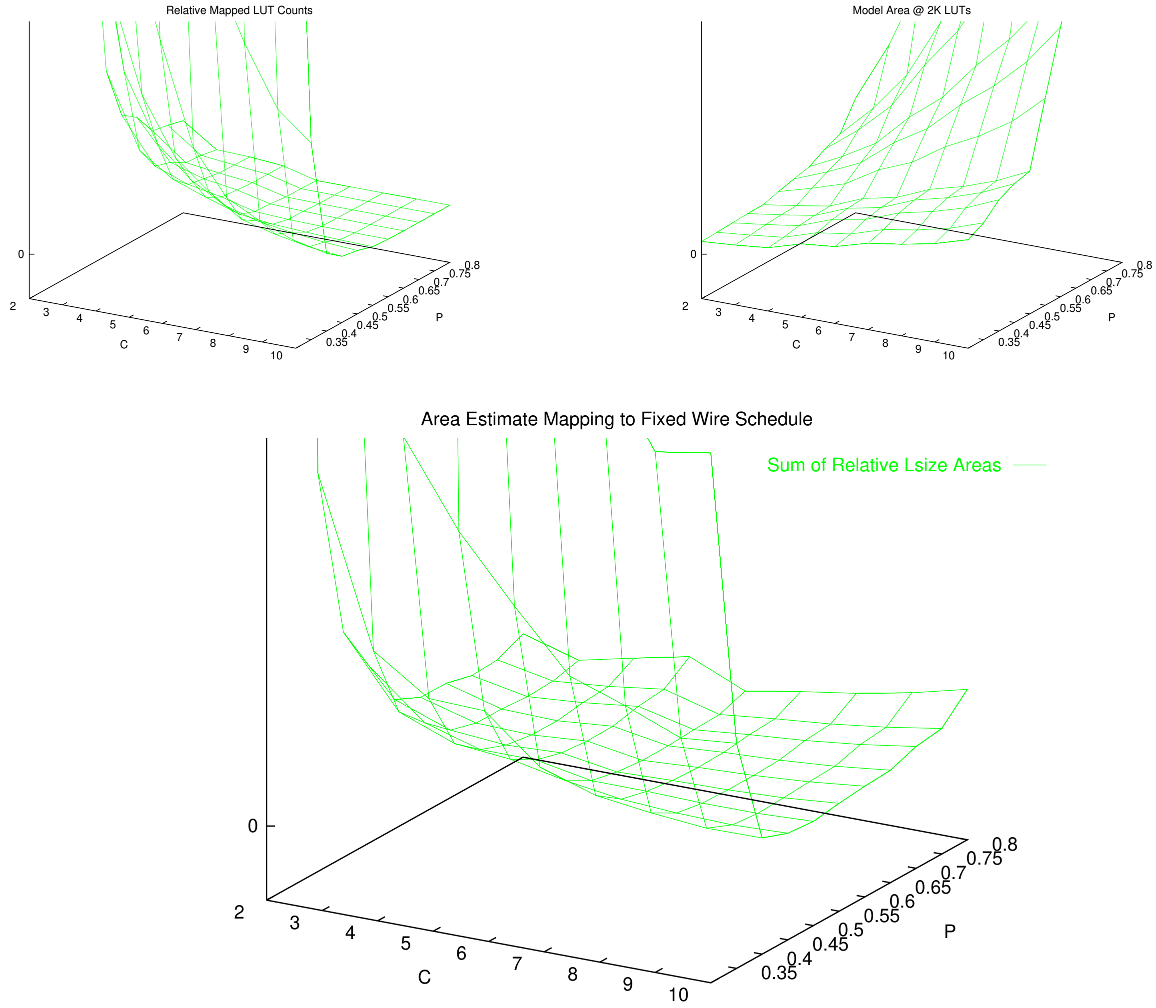


Figure 15: Area Utilization Results Mapping Benchmark to Fixed Wire Schedules

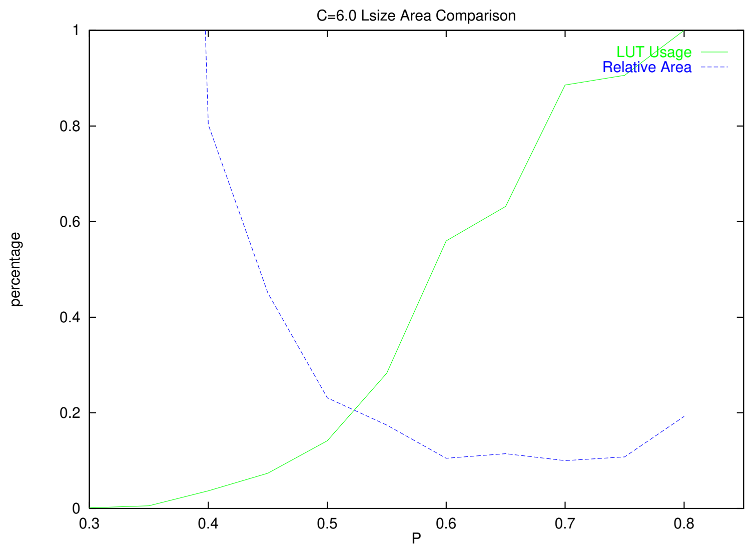


Figure 16: p-space slice for `i10` showing that area minimization is not directly correlated with high LUT usage

Wire Dominated $WP = 8\lambda$, $A_{sw} = 64\lambda^2$

Minimization Objective	params C	params P	Sigma relative area	Max relative area	LUT Utilization
relative area	6	0.6	1.30	2.99	0.87
max relative area	6	0.65	1.40	1.91	0.93
area with full utilization	10	0.75	3.23	6.94	1.00

Linear $WP = 8\lambda$, $A_{sw} = 2500\lambda^2$

Minimization Objective	params C	params P	Sigma relative area	Max relative area	LUT Utilization
relative area	6	0.6	1.23	2.84	0.87
max relative area	6	0.65	1.24	2.38	0.89
area with full utilization	10	0.75	2.98	4.87	1.00

Switch Dominated (Quadratic)
$WP = 8\lambda$, $A_{sw} = 2500\lambda^2$

Minimization Objective	params C	params P	Sigma relative area	Max relative area	LUT Utilization
relative area	6	0.6	1.32	3.50	0.87
max relative area	4	0.65	1.47	2.31	0.49
area with full utilization	10	0.75	4.25	11.5	1.00

Table 1: Compare Effects of Various Network Selection Points

area, while the full utilization designs stay at roughly the same point.

8 Limitations and Future Study

We have only scratched the surface here. As with any CAD effort where we are solving NP-hard problems with heuristic solutions there is a significant tool bias to the results. Flowmap was not attempting to minimize interconnect requirements, and there is a good argument that LUT covering and fixed-wire schedule partitioning should be considered together to get the best results. At the very least, it would be worthwhile to try different LUT mapping strategies to assess how much these results are effected by LUT covering.

The area model used assumes a purely hierarchical, 2-ary interconnect. Two things one would like to explore are (1) the effects of different arity (flattening the tree) and (2) the introduction of shortcut connections (*e.g.* Fat Pyramid [9]). The shortcut connections will tend to reduce the need for bandwidth in the root channel and may shift the balance in interconnect costs. Further, shortcuts appear essential for delay-mapped designs, which we have also not studied here.

We suspect the hierarchical model captures the high-level requirements of any network, but it will be interesting to study these effects more specifically for mesh-based architectures. The key algorithmic enabler needed for both shortcuts and mesh-based architectures is to identify good heuristics for spreading in two dimensions rather than the one-dimensional approach we exploited here.

An important assumption we have made here is that interconnect growth is geometric (power law). The c, p estimates shown in Figures 9 and 11 support the fact that a geometric growth relationship seems fairly reasonable. Nonetheless, we have not directly explored wire-schedules which deviate from strict geometric growth, and there may be better schedules to be found outside of the strict geometric growth space explored here.

We concentrated on global wiring requirements here and have not focussed on detailed switch population. The robustness of the general trends across different area and population models shown in Table 1 suggests that the major effects identified here are independent of the switch population details. While this does show us the relative merits of a given interconnect richness within a particular population model, we cannot, however, make any conclusions about the relative merits of different population schemes without carefully accounting for detailed population effects in both the area model and routability assessment.

9 Conclusions

We see that wires and interconnect are the dominant area components of FPGA devices. We also see that the amount of interconnect needed per LUT varies both among designs and within a single design. Given that this is the case, we cannot use all of our LUTs and all of our interconnect to their full potential all of the time—we must underutilize one resource in order to fully utilize the other. If we focus on LUT utilization, we waste significant interconnect—our dominant area resource. This suggests, instead, it may be more worthwhile for us to focus on interconnect utilization even if it means letting some LUTs go unused. Answering our opening question, we see that higher LUT usage does not imply lower area and that LUT usability is not always directly correlated with area efficiency.

Acknowledgements

This research is part of the Berkeley Reconfigurable Architectures Software and Systems effort supported by the Defense Advanced Research Projects Agency under contract numbers F30602-94-C-0252 and DABT63-C-0048 and directed by Prof. John Wawrzynek and the author.

Jason Cong's VLSI CAD Lab at UCLA provided the Flowmap implementation used to map LUTs here. Kip Macy did the actual benchmark mapping to LUTs and developed the initial BLIF parser used for these experiments.

Feedback from Randy Huang, Nicholas Weaver, John Wawrzynek, and Eylon Caspi helped cleanup early drafts of this paper.

References

[1] Aditya A. Agarwal and David Lewis. Routing Architectures for Hierarchical Field Programmable Gate Arrays. In *Proceedings 1994 IEEE International Conference on Computer Design*, pages 475–478. IEEE, October 1994.

[2] Rick Amerson, Richard Carter, W. Bruce Culbertson, Phil Kuekes, and Greg Snider. Plasma: An FPGA for Million Gate Systems. In *Proceedings of the International Symposium on Field Programmable Gate Arrays*, pages 10–16, February 1996.

[3] Sandeep Bhatt and Frank Thomson Leighton. A Framework for Solving VLSI Graph Layout Problems. *Journal of Computer System Sciences*, 28:300–343, 1984.

[4] Gaetano Borriello, Carl Ebeling, Scott Hauck, and Steven Burns. The Triptych FPGA Architecture. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 3(4):491–501, December 1995.

[5] Stephen D. Brown, Robert J. Francis, Jonathan Rose, and Zvonko G. Vranesic. *Field-Programmable Gate Arrays*. Kluwer Academic Publishers, 101 Philip Drive, Assinippi Park, Norwell, Massachusetts, 02061 USA, 1992.

[6] Jason Cong and Yuzheng Ding. FlowMap: An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-Table Based FPGA Designs. *IEEE Transactions on Computer-Aided Design*, 13(1):1–12, January 1994.

[7] André DeHon. Entropy, Counting, and Programmable Interconnect. In *Proceedings of the 1996 International Symposium on Field Programmable Gate Arrays*. ACM/SIGDA, February 1996. Extended version available as Transit Note #128 `<http://www.ai.mit.edu/projects/transit/transit-notes/tn128.ps.Z>`.

[8] André DeHon. Reconfigurable Architectures for General-Purpose Computing. AI Technical Report 1586, MIT Artificial Intelligence Laboratory, 545 Technology Sq., Cambridge, MA 02139, October 1996. Anonymous FTP `publications.ai.mit.edu:ai-publications/1996/AITR-1586.ps.Z`.

[9] Ronald Greenberg. The Fat-Pyramid and Universal Parallel Computation Independent of Wire Delay. *IEEE Transactions on Computers*, 43(12):1358–1365, December 1994.

[10] Kenneth M. Hall. An r-dimensional Quadratic Placement Algorithm. *Managment Science*, 17(3):219–229, November 1970.

[11] Yen-Tai Lai and Ping-Tsung Wang. Hierarchical Interconnect Structures for Field Programmable Gate Arrays. *IEEE Transactions on VLSI Systems*, 5(2):186–196, June 1997.

[12] B. S. Landman and R. L. Russo. On Pin Versus Block Relationship for Partitions of Logic Circuits. *IEEE Transactions on Computers*, 20:1469–1479, 1971.

[13] Frank Thomson Leighton. New lower bound techniques for VLSI. In *Twethy-Second Annual Symposium on the Foundations of Computer Science*. IEEE, 1981.

[14] Charles E. Leiserson. Fat-Trees: Universal Networks for Hardware Efficient Supercomputing. *IEEE Transactions on Computers*, C-34(10):892–901, October 1985.

[15] Jonathan Rose and Stephen Brown. Flexibility of Interconnection Structures for Field-Programmable Gate Arrays. *IEEE Journal of Solid-State Circuits*, 26(3):277–282, March 1991.

[16] Ellen M. Sentovich, Kanwar Jit Singh, Luciano Lavagno, Cho Moon, Rajeev Murgai, Alexander Saldanha, Hamid Savoj, Paul R. Stephan, Robert K. Brayton, and Alberto Sangiovanni-Vincentelli. SIS: A System for Sequential Circuit Synthesis. UCB/ERL M92/41, University of California, Berkeley, May 1992.

[17] Atsushi Takahara, Toshiaki Miyazaki, Takahiro Murooka, Masaru Katayama, Kazuhiro Hayashi, Akihiro Tsutsui, Takaki Ichimori, and Ken nosuke Fukami. More Wires and Fewer LUTs: A Design Methodology for FPGAs. In *Proceedings of the 1998 International Symposium on Field-Programmable Gate Arrays*, pages 12–19, 1998.

A Mapped Benchmarks Statistics used for Experiment

Design	Mapped LUTs	Max c	Max p	Avg. c	Avg. p
5xp1	83	7	0.35	6	0.42
9sym	177	6	0.57	5	0.63
9symml	80	6	0.45	5	0.47
C1355	74	7	0.64	5	0.67
C17	2	0	0	0	0
C1908	136	7	0.56	5	0.56
C2670	218	6	0.7	4	0.75
C3540	382	7	0.54	6	0.56
C432	79	7	0.53	6	0.57
C499	74	7	0.63	5	0.67
C5315	590	7	0.66	5	0.69
C6288	522	8	0.44	6	0.47
C7552	723	7	0.63	5	0.68
C880	116	6	0.65	5	0.65
a	1	0	0	0	0
alu2	279	8	0.49	6	0.54
apex1	799	8	0.62	7	0.6
apex3	900	9	0.54	6	0.58
apex6	258	6	0.71	5	0.68
apex7	108	6	0.61	4	0.64
b1	4	4	0.40	3	0.67
b12	377	7	0.55	7	0.46
b9	56	6	0.62	4	0.69
bbara	34	6	0.43	5	0.44
bbsse	70	8	0.32	6	0.36
bbtas	9	5	0.25	4	0.3
beecount	21	6	0.43	5	0.49
c8	87	7	0.54	5	0.58
cc	33	5	0.65	4	0.73
cht	68	5	0.71	4	0.69
clip	243	7	0.53	6	0.44
clmb	369	2	1	2	1
cm138a	9	5	0.44	5	0.48
cm150a	15	5	0.67	4	0.75
cm151a	8	5	0.53	4	0.63
cm152a	11	5	0.41	5	0.43
cm162a	14	6	0.49	5	0.56
cm163a	12	5	0.60	5	0.65
cm42a	10	5	0.45	5	0.45
cm82a	4	4	0.5	4	0.5
cm85a	12	5	0.42	5	0.44
cmb	19	6	0.49	5	0.53
comp	45	6	0.52	5	0.57
con1	6	5	0.31	5	0.33
count	37	6	0.58	4	0.64
cps	821	9	0.62	6	0.67
cse	134	6	0.57	5	0.58
cu	24	6	0.51	5	0.56
daio	3	0	0	0	0
dalu	502	8	0.55	6	0.59
decod	18	5	0.49	4	0.52
dk14	51	7	0.35	7	0.24
dk15	31	6	0.39	6	0.22
dk16	171	8	0.42	6	0.47
dk17	30	6	0.38	5	0.41
dk27	5	5	0.15	4	0.35
dk512	25	6	0.17	5	0.22
donfile	1	0	0	0	0
duke2	274	8	0.48	6	0.53
e64	386	7	0.6	5	0.64
ex1	164	6	0.6	5	0.64
ex2	90	6	0.5	5	0.56
ex3	33	6	0.45	5	0.52
ex4	41	7	0.36	6	0.42
ex5	27	6	0.36	5	0.41
ex6	71	6	0.51	6	0.4
ex7	36	6	0.51	5	0.54
example2	139	6	0.69	4	0.74
f51m	98	7	0.53	5	0.58
frg1	282	6	0.62	6	0.51
frg2	744	8	0.61	5	0.65
i1	19	5	0.66	4	0.74
i10	906	7	0.68	5	0.71
i2	77	6	0.82	5	0.86
i3	46	5	0.88	5	0.89
i4	97	5	0.77	4	0.82
i5	153	6	0.67	4	0.75
i6	144	5	0.76	4	0.78
i7	215	6	0.71	4	0.77
i9	347	7	0.63	5	0.65
keyb	209	8	0.43	6	0.48
kirkman	133	8	0.35	6	0.37
lal	67	7	0.56	5	0.6
ldd	50	7	0.46	5	0.5
lion	3	0	0	0	0
lion9	5	0	0	5	0.1
majority	4	5	0.13	5	0.17
mark1	52	7	0.41	4	0.66
mc	9	5	0.21	4	0.33
misex1	24	6	0.37	5	0.41
misex2	54	6	0.57	5	0.59
mm30a	327	6	0.57	6	0.47
mm4a	118	9	0.28	7	0.31
mm9a	96	6	0.48	6	0.38
mm9b	120	6	0.51	5	0.53
modulo12	1	0	0	0	0
mult16a	32	5	0.36	4	0.36
mult16b	31	5	0.25	4	0.16
mult32a	64	5	0.39	4	0.38
mult32b	62	6	0.41	5	0.43
mux	45	6	0.51	6	0.47
my_adder	32	4	0.61	4	0.61
o64	46	5	0.84	4	0.88
opus	50	6	0.51	5	0.57
pair	567	8	0.6	5	0.65
parity	5	5	0.73	5	0.75
pcle	23	5	0.5	4	0.59
pcler8	33	5	0.6	4	0.68
planet	410	9	0.45	6	0.5
planet1	410	9	0.45	6	0.5
pm1	28	6	0.54	4	0.61
rd53	36	6	0.38	5	0.4
rd73	190	6	0.56	5	0.57
rd84	405	7	0.63	5	0.67
rot	467	7	0.65	5	0.71
s1	317	8	0.52	6	0.56
s1196	226	7	0.53	6	0.45
s1238	253	7	0.52	7	0.46
s1423	162	6	0.51	5	0.40
s1488	289	7	0.55	4	0.76
s1494	295	7	0.59	6	0.51
s1a	6	1	1	1	1
s208	27	6	0.34	5	0.29
s27	6	5	0.20	0	0
s298	40	6	0.35	5	0.41
s344	41	6	0.34	4	0.44
s349	41	6	0.34	4	0.44
s382	53	6	0.44	5	0.48
s386	74	7	0.32	6	0.36
s400	53	6	0.38	5	0.42
s420	62	7	0.31	5	0.39
s444	53	6	0.37	5	0.39
s510	105	7	0.45	6	0.47
s526	84	7	0.39	5	0.41
s526n	90	7	0.39	5	0.42
s5378	522	7	0.63	5	0.66
s641	79	5	0.6	4	0.68
s713	80	6	0.61	5	0.67
s8	1	0	0	0	0
s820	166	7	0.48	6	0.52
s832	169	7	0.48	6	0.52
s838	124	7	0.44	5	0.5
s9234	439	8	0.56	5	0.63
s953	182	7	0.5	6	0.54
sand	406	7	0.57	5	0.61
sao2	121	7	0.47	6	0.5
sbc	332	7	0.57	5	0.6
scf	663	9	0.53	6	0.57
sct	69	7	0.48	5	0.52
shiftreg	2	0	0	0	0
sqrt8	45	7	0.46	6	0.37
sqrt8ml	40	6	0.58	5	0.47
squar5	50	8	0.27	6	0.34
sse	70	8	0.32	6	0.36
styr	341	7	0.59	7	0.47
t481	735	8	0.55	6	0.6
table3	513	9	0.52	7	0.55
table5	522	7	0.66	7	0.53
tav	12	5	0.23	4	0.50
tbk	616	9	0.5	6	0.54
tcon	16	4	0.78	3	0.88
term1	246	8	0.48	6	0.52
train11	19	6	0.48	4	0.76
train4	3	0	0	0	0
ttt2	198	9	0.42	6	0.45
unreg	32	5	0.67	5	0.7
vda	517	8	0.55	6	0.58
vg2	277	9	0.42	6	0.47
x1	761	9	0.48	6	0.57
x2	23	6	0.39	5	0.53
x3	441	7	0.61	5	0.6
x4	294	7	0.61	5	0.63
xor5	21	6	0.38	5	0.46
z4ml	80	7	0.39	5	0.41
alu4	1756	8	0.58	6	0.63
apex4	1284	7	0.69	5	0.72
apex5	1241	9	0.63	6	0.66
cordic	1381	9	0.62	8	0.52
dsip	1175	7	0.65	6	0.64
ex5p	1348	9	0.61	6	0.64
i8	1242	8	0.57	5	0.60
k2	1138	8	0.69	6	0.58
seq	2004	10	0.53	7	0.58

B Lsize and Level

When mapping to a hierarchical array, or any array for that matter, one problem to address is how we count area used. Do we charge the design for the smallest tree hierarchy used? If so, we only get a logarithmic estimation of size. Designs which are slightly larger than a tree stage are charged the full cost of the next tree level. This could skew measures as ± 1 LUT at a power-of-two boundary has a big difference in metric, but elsewhere near factor-of-two differences hardly matter. For the data shown here, we have counted size in terms of the span of LUTs used (*lsize* — See adjacent diagram). That is, if we number the tree LUTs in a linear order; we pack starting at LUT 0 and use the position of the highest placed LUT to account for the capacity used. The LUTs above the last used subtree are all free. Intuitively, if we consume all of a subtree of size 128 and one more subtree of size 64, we still have a subtree of size 64 available for additional logic, so we charge the design to be only of lsize 192. In practice, when we use level as a metric instead of lsize, we see similar trends to those reported here but a larger benchmark-wide mismatch penalty, especially when requiring full population, due to the logarithmic granularity effects.