

Design of FPGA Interconnect for Multilevel Metallization

André DeHon, *Member, IEEE*, and Raphael Rubin

Abstract—How does multilevel metallization impact the design of field-programmable gate arrays (FPGA) interconnect? The availability of a growing number of metal layers presents the opportunity to use wiring in the third dimension to reduce area and switch requirements. Unfortunately, traditional FPGA wiring schemes are not designed to exploit these additional metal layers. We introduce an alternate topology, based on Leighton’s mesh-of-trees (MoT), which carefully exploits hierarchy to allow additional metal layers to support arbitrary device scaling. When wiring layers grow sufficiently fast with aggregate network size (N), our network requires only $O(N)$ area; this is in stark contrast to traditional, Manhattan FPGA routing schemes where switching requirements alone grow superlinearly in N . In practice, we show that, even for the admittedly small designs in the Toronto “FPGA Place and Route Challenge,” arity-4 MoT networks require 26% fewer switches than the standard, Manhattan FPGA routing scheme.

Index Terms—Field programmable gate array (FPGA), hierarchical, interconnect, mesh-of-trees (MoT), multilevel metallization, Rent’s rule.

I. INTRODUCTION

VLSI technology has advanced considerably since the first field-programmable gate arrays (FPGAs) [1]. Feature sizes have shrunk, die sizes and raw capacities have grown, and the number of metal layers available for interconnect has grown. The most advanced VLSI processes now sport 7–9 metal layers, and metal layers have grown roughly logarithmically in device capacity [2].

How should this shift in available resources affect the way we design FPGAs?

One can view multilevel metallization, and particularly, the current rate of scaling, as an answer to the quandary that interconnect requirements for typical designs [Rent’s Rule [3], $p > 0.5$; see (7)] grows faster than linearly with gate count [4], [5]. If we can accommodate the growing wire requirements in the third-dimension using multiple wire layers, then we may be able to maintain constant density for our devices. Alternately, if we cannot do this, the two-dimensional (2-D) density of our devices necessarily decreases as we go to larger device capacities.

The existence of additional metal layers is not sufficient, by itself, to stave off this problem. We must further guarantee that we can contain the active silicon area to a bounded area per

device (*e.g.*, an asymptotically constant number of switches per gate) and that we can topologically arrange to use the additional metallization.

We show that the dominant, traditional, Manhattan style, interconnect scheme is not directly suited to exploiting multilevel metallization (Section II). Its superlinear switch requirements preclude it from taking full advantage of additional metal layers. The density of these architectures ultimately decreases with increasing gate count.

We introduce an alternative topology, based on Leighton’s mesh-of-trees [6], [7] (MoT), which exploits hierarchy more strictly while retaining the 2-D interconnect style of the Manhattan interconnect (Section III). We show that this topology has an asymptotically constant number of switches per endpoint assuming no domain coloring limitations (Section IV-B). We further show that it can be arranged to fully exploit additional metal layers. As a result, given a sufficient interconnect layer growth rate, the gate density remains constant across increasing gate counts.

In Section IV, we summarize a set of empirical experiments to compare the switch and wiring requirements of our MoT design to standard Manhattan routing topologies and characterize growth effects. In Section V, we explore several variations in the detail design of the MoT and identify versions which provide 26% fewer switches than the best known Manhattan designs. Table I summarizes the symbols used throughout the article.

II. MANHATTAN INTERCONNECT

A. Base Model

Fig. 1 shows the standard model of a Manhattan (symmetric) [8], interconnect scheme. Each compute block (look-up table (LUT) or island of LUTs) is connected to the adjacent channels by a C-box. At each channel intersection is a switchbox (S-box). In the C-box, each compute block IO pin is connected to a fraction of the wires in a channel. At the S-box, each channel on each of the four sides of the S-box connects to one or more channels on the other sides of the S-box.

Early experiments [8] studied the number of sides of the compute block on which each input or output of a gate appeared (T), the fraction of wires in each channel each of these signals connected to (F_c), and the number of switches connected to each wire entering an S-box (F_s). Regardless of the detail choices for these numbers, they have generally been considered constants, and the asymptotic characteristics are independent of the particular constants chosen.

To keep this general, we will simply assume each side of the compute block has I inputs or outputs to the channel. If

Manuscript received September 10, 2003; revised January 3, 2004. This research was funded in part by the Defense Advanced Research Projects Agency (DARPA) Moletronics Program under Grant ONR N00014-01-0651 and in part by the National Science Foundation CAREER program under Grant CCR-0133102.

The authors are with the California Institute of Technology, Pasadena, CA 91125 USA (e-mail: andre@cs.caltech.edu).

Digital Object Identifier 10.1109/TVLSI.2004.827562

TABLE I
 SUMMARY OF SYMBOLS USED IN ARTICLE

Symbol	Description	Intro
a	Tree arity	Sec. V-A & Fig. 10
B_{sw}	Total switches per node	Eq. 3, 20, 26
BW	Bisection width	Sec. II-A
C	Trees per row and column in MoT	Sec. III-A
C_{lb}	Lower bound on C	Eq. 28
c_r	Constant in Rent's Rule	Eq. 7
C_{sw}	C-box switches per node	Eq. 1, 15
F_c	Fraction of channels connected to each input/output in C-box	Sec. II-A
F_s	Switches per wire in S-box	Sec. II-A
g_i	Growth rate at level i in tree	Sec. V-A.1
HV_{sw}	Switches connecting horizontal and vertical trees per node	Eq. 16, 17
I	Total inputs/outputs from a node to each adjacent channel	Sec. II-A & Fig. 1
IO	Total input/output from a region	Sec. II-A
k	Inputs to logic block	Sec. II-A
L	Number of wire layers	Sec. II-D
l	Number of levels in tree	Sec. II-C, V-A
l_{row}	length of a row in nodes	Sec. III-B
L_{seg}	Segment length	Sec. II-B
N	nodes/compute blocks in network	Sec. II-A
N_{sw}	Total number of switches in a device	Eq. 13
p	Growth exponent in Rent's Rule	Eq. 7
r	Ratio of switch width to wire pitch	Sec. III-C.1
S_{sw}	S-box switches per node	Eq. 2, 10
T	Number of sides on which each input/output of a node appears	Sec. II-A & Fig. 8
T_{sw}	Tree switches per base channel per direction per node	Eq. 18, 25, 40
T_{sw_short}	Tree switches per base channel with shortcuts	Eq. 42
W	Channel width	Sec. II-A & Fig. 1
W_b	Base channel width for hierarchical segmentation	Sec. II-C
w_{ch}	Wire tracks in a single tree at a given level	Eq. 22, 30
W_x	Total channel wires across all hierarchical levels	Eq. 12, 27 & Sec. V-A.5
w_x	Total wire tracks in a single tree across all hierarchical levels	Sec. V-A.3

we are thinking about a single-output, k -input LUT (k -LUT) as our compute block, then $I = (T \times (k + 1))/4$. The number of switches in a C-box is

$$C_{sw} = 2 \cdot F_c \cdot I \cdot W \quad (1)$$

where W is the width of the channel. Each S-box requires

$$S_{sw} = \left(\frac{4}{2}\right) \cdot F_s \cdot W = 2 \cdot F_s \cdot W. \quad (2)$$

Each compute block comes with two C-boxes and one S-box (as shown highlighted in Fig. 1). So, the total number of switches per compute block is

$$B_{sw} = 2 \cdot C_{sw} + S_{sw} = 2W(2 \cdot F_c \cdot I + F_s). \quad (3)$$

Dropping the constants we get

$$B_{sw} = O(W). \quad (4)$$

That is, we see that the number of switches required per compute block is linear in W , the channel width.

We can get a loose bound on channel width simply by looking at the bisection width of the design. If a design has a minimum bisection width BW , then we have a lower bound on the channel width

$$BW \leq \sqrt{N} \cdot W. \quad (5)$$

That is, we must provide at least BW wires across the \sqrt{N} row (or column) channels which cross the middle of a chip with N nodes. This allows us to solve for a lower bound on W

$$W \geq \frac{BW}{\sqrt{N}}. \quad (6)$$

Empirically, we find that the bisection width of a design can often be characterized by the Rent's Rule relation [3]¹

$$BW = IO = c_r N^p. \quad (7)$$

This now allows us to define a correspondence between W and N

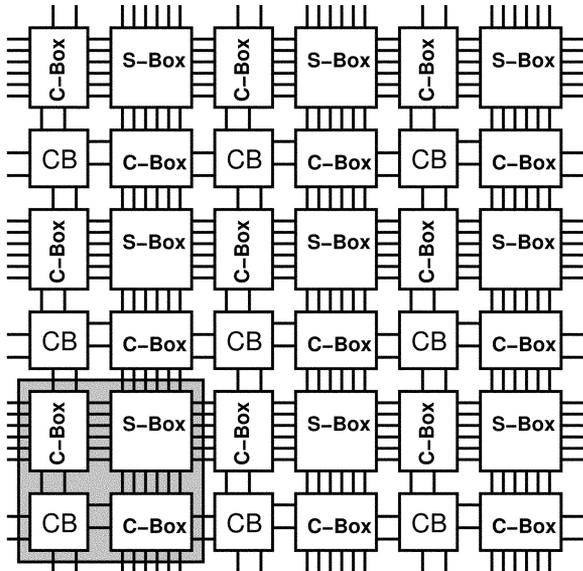
$$W \geq \frac{c_r N^p}{\sqrt{N}} = \Omega\left(N^{(p-0.5)}\right). \quad (8)$$

This is the same correspondence which one gets by combining the results of Donath [10] and El Gamal [11] for $p > 0.5$. This means

$$B_{sw} = O(W(N)) = \Omega\left(N^{(p-0.5)}\right). \quad (9)$$

All together, this says that as we build larger designs, if the interconnect richness is greater than $p = 0.5$, the switch requirement per compute block is growing for the Manhattan topology;

¹We might say the bisection width is determined by the IO out of each half, so $BW = IO(N/2)$, rather than $BW = IO(N)$, but this would only make a constant factor difference and not change our asymptotic conclusions.


 Fig. 1. Manhattan interconnect model ($W = 6, I = 2$ shown).

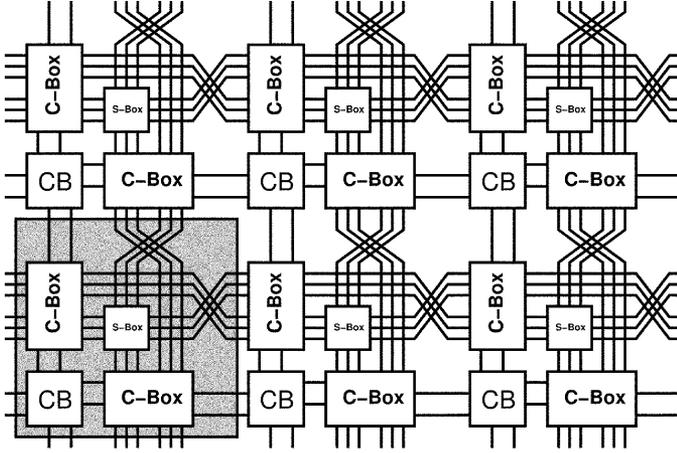


Fig. 2. Segmentation in Manhattan interconnect model (example shows $L_{\text{seg}} = 2$).

this means the aggregate switching requirements grow superlinearly with the number of compute blocks supported. Regardless of the metallization offered, our designs will decrease in density with increasing gate count.

B. Segmentation

Modern designs, both in practice and in academic studies use segments which span more than one switchbox (see Fig. 2). For example, a recent result from Betz suggests that length 4–8 buffered segments require less area than alternatives [12]. The important thing to notice is that any **fixed** segmentation scheme only changes the constants and not the asymptotic growth factor in (9). In particular, using a single segmentation scheme of length L_{seg} will change (2) to

$$S_{\text{sw}} = \left(\frac{1}{L_{\text{seg}}} \right) (2) F_s \cdot W = \left(\frac{2}{L_{\text{seg}}} \right) F_s \cdot W. \quad (10)$$

In practice the W will be different between the segmented and nonsegmented cases, with the segmented cases requiring larger W 's, but the asymptotic lower bound relationship on W derived above still holds. Similarly, a mixed segmentation scheme will also change the constants, but not the asymptotic requirements.

C. Hierarchical

A strictly hierarchical segmentation scheme might allow us to reduce the switchbox switches. Consider, that we have a base number of wire channels W_b , and populate the channel with W_b single length segments, W_b length 2 segments, W_b length 4 segments, and so forth. Using (10) with W_b in for W and summing across the geometric wire lengths, we see the total number of switches needed per switchbox is

$$\begin{aligned} S_{\text{sw}} &= \left(\sum_{L_{\text{seg}}=1}^l \left(\frac{1}{L_{\text{seg}}} \right) \right) (2) F_s \cdot W_b \\ &= \left(\frac{1}{1} + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \dots \right) \cdot 2 \cdot F_s \cdot W_b \\ &\leq 4 \cdot F_s \cdot W_b \end{aligned} \quad (11)$$

where l is the total number of hierarchical levels used. The total number of wires in a channel, W_x , is now

$$W_x = l \cdot W_b. \quad (12)$$

For sufficiently large l , we can raise W_x to the required bisection width. Since S_{sw} in this hierarchical case does not, asymptotically, depend on l , the number of switches converges to a constant.

However, we should note this still does not change the asymptotic switch requirements, since the switch requirements depend on both the C-box switches and the S-box switches. As long as the C-box switches continue to connect to a constant fraction of W and not W_b , the C-box contribution to the total number of switches per compute block (1) continues to make the total number of switches linear in W and hence growing with N .

From this we see clearly that it is the flat connection of block IO's to the channel which ultimately impedes scalability.

D. Switch Dominated

Conventional experience implementing this style of interconnect has led some people to observe that switch requirements tend to be limiting rather than wire requirements (*e.g.*, [12]). Asymptotically, we see that an N -node FPGA will need

$$N_{\text{switch}}(N) = B_{\text{sw}} \cdot N = \Omega \left(N^{(p+0.5)} \right). \quad (13)$$

With BW wires in the bisection and a number of wire layers, L , we will require at least

$$A_{\text{wire}}(N) \geq \left(\frac{BW}{L} \right)^2 = \Omega \left(\frac{N^{2p}}{L^2} \right). \quad (14)$$

For fixed wire layers, this says wiring requirements grow slightly faster than switches (*i.e.*, when $p > 0.5$, $2p > p + 0.5$). Asymptotically, this suggests that the number of layers must grow at least as fast as $\Omega(N^{(\frac{2p-1}{4})})$ in order for the design to remain switch dominated. Since switches have a much larger constant contribution than wires, it is not surprising that designs require a large N before these asymptotic effects become apparent.

III. MESH OF TREES

The asymptotic analysis in Section II says that it is necessary to bound the compute block connections to a constant if we hope to contain the total switches per compute block to a constant independent of design size. Leighton's MoT network [6], [7] is a topology which does just that. Simply containing the switches to a constant is necessary but not sufficient to exploit additional metal layers. Later in this section, we also show that the MoT topology can be wired within a constant layout area per compute block.

A. Basic Arrangement

In the MoT arrangement, we build a tree along each row and column of the grid of compute elements (see Fig. 3). For now,

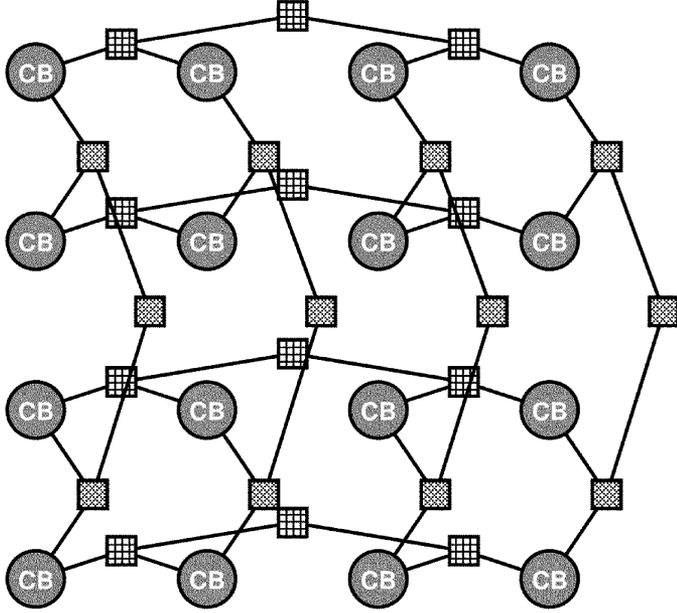


Fig. 3. Basic MoT topology.

we will assume the tree is binary (arity 2), but we can certainly vary the arity of the tree as one of the design parameters (Section V-A). The compute blocks connect only to the lowest level of the tree. A connection can then climb the tree in order to get to longer segments. We can place multiple such trees along each row or column to increase the routing capacity of the network. Each compute block is simply connected to the leaves of the set of horizontal and vertical trees which land at its site. We can parameterize the way the compute block connects to the leaf channels in a manner similar to the Manhattan C-box connections above.

We will use the parameter C to denote the number of trees which we use in each row and column. A MoT row or column tree with $C = 2$ is shown in Fig. 12(b). The C-box connections at each “channel” in this topology are made only to the C wires which exist at the leaf of the tree

$$C_{\text{sw}} = 2F_c \cdot I \cdot C. \quad (15)$$

In the simplest sense, we do not have switch boxes in this topology. At the leaf level, we allow connections between horizontal and vertical trees. Typically, we consider allowing each horizontal channel to connect to a single vertical channel in a domain style similar to that used in typical Manhattan switchboxes. This gives

$$HV_{\text{sw}} = C. \quad (16)$$

It would also be possible to fully populate this corner turn, allowing any horizontal tree to connect to any vertical tree at points of leaf intersection without changing the asymptotic switch requirements

$$HV_{\text{sw}} = C^2. \quad (17)$$

Within each row or column tree, we need a switch to connect each lower channel to its parent channel. This can be as simple

as a single pass transistor and associated memory cell. Amortizing across the compute blocks which share a single tree, per compute block we need a total of

$$T_{\text{sw}} = 1 + \frac{1}{2} + \frac{1}{4} + \dots < 2. \quad (18)$$

The horizontal channel holds C such trees, as does the vertical channel. Thus, each compute block needs

$$\begin{aligned} B_{\text{sw}} &= C_{\text{sw}} + HV_{\text{sw}} + 2 \cdot C \cdot T_{\text{sw}} \\ &< C_{\text{sw}} + HV_{\text{sw}} + 4 \cdot C. \end{aligned} \quad (19)$$

Using the linear corner turn population (16)

$$\begin{aligned} B_{\text{sw}} &< 2 \cdot F_c \cdot I \cdot C + 5 \cdot C \\ &< (2 \cdot F_c \cdot I + 5) \cdot C. \end{aligned} \quad (20)$$

Assuming we can hold C bounded with increasing design size, this leaves us with a constant number of switches per compute block.

B. Tree Growth

The strict binary tree we have shown corresponds to $p = 0.5$. To accommodate larger p values, it is necessary to grow the number of parents in the tree. Returning to (8), we need the per channel root bandwidth to be $CN^{(p-0.5)}$. We can arrange to support a larger p with the MoT by increasing the stage-to-stage growth rate.

For example, if alternate tree levels double the number of parent segments, we can achieve $p = 0.75$ (see Fig. 4). The number of tree levels is

$$l = \log_2 \left(l_{\text{row}} = \sqrt{N} \right) \quad (21)$$

where l_{row} is the length of each row or column. The number of channels, w_{ch} , composing the root level, l , of each tree will thus be

$$w_{\text{ch}}(l) = 2^{(l/2)} = 2^{(\log_2(\sqrt{N})/2)} = 2^{(\log_2(\sqrt[4]{N}))} = \sqrt[4]{N}. \quad (22)$$

The total bisection width at this level is the aggregate channel capacity across all \sqrt{N} channels across the chip

$$\text{BW} = \sqrt{N} \cdot w_{\text{ch}}(l). \quad (23)$$

In this case that becomes

$$\text{BW} = \sqrt{N} \cdot \sqrt[4]{N} = N^{0.75}. \quad (24)$$

That is, this growth is equivalent to providing $p = 0.75$.

Fig. 4 shows one-dimensional (1-D) slices of a $p = 0.75$ MoT (top) and a flat Manhattan topology (bottom). The MoT accommodates the bisection width of 4 using only a single base domain ($C = 1$), while the Manhattan topology requires at least one domain for every wire in the bisection; this demonstrates how the MoT can often get away with a smaller C than the Manhattan channel width (W). The total channel wires for the MoT is 13 ($W_x = 13$). Asymptotically, the MoT will require six switches per endpoint for this arrangement, while the Manhattan design requires 8 to accommodate this channel width of 4. For larger

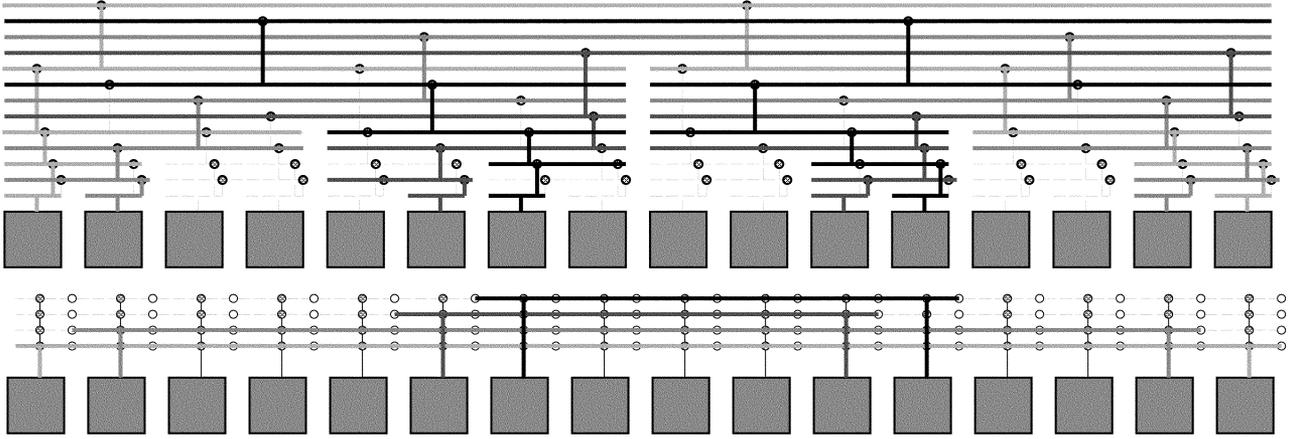


Fig. 4. Row/column tree growth to achieve $p = 0.75$ and bisection width comparison with mesh.

TABLE II
SWITCHES/NODE FOR $p = 0.75$ MoT

Leaf Span	Wires	Switches per Wire	Switches per Endpoint
1	1	2	2
2	2	1	1
4	2	2	1
8	4	1	1/2
16	4	2	1/2
32	8	1	1/4
64	8	2	1/4
⋮	⋮	⋮	⋮

spans, the effect increases. For a span of 32 nodes, the MoT can accommodate a bisection bandwidth of 8 while still using at most six switches per endpoint; the mesh with a bisection width of 8 will require 16 switches per endpoint.

Note that even though we increased the rate of wire growth, the total number of switches per node remain asymptotically constant (see Table II)

$$T_{\text{sw}} = 2 + 1 + 1 + \frac{1}{2} + \frac{1}{2} + \frac{1}{4} + \frac{1}{4} \cdots < 6. \quad (25)$$

Which makes

$$B_{\text{sw}}(p = 0.75) < (2 \cdot F_c \cdot I + 13) \cdot C. \quad (26)$$

This property holds for any $p < 1.0$. That is, given sufficiently large N , we can approximate any p by programming the stage-to-stage growth rate, and the number of switches per compute block remains asymptotically constant. The particular constant grows with p as this example suggests and is developed further in Section V-A. For arbitrary design bisection width, we can pick a p ($p > 0.5$) that is equal to or greater than the design p , and a network with constant switches per endpoint can provide that much bisection bandwidth.

We are thus able to satisfy the lower bound relationship (8) introduced in Section II with constant switches per compute block. However, the lower bound relationship only guarantees that we have sufficient wires in the bisection, *if we can use them*. The population scheme will determine whether or not enough of the wires can be used to keep C bound to a constant. At this point,

we have no proof of the sufficiency of the population, so we employ empirical experiments, reported in Section IV, to assess the sufficiency of this population scheme.

C. Basic Layout

Constant switches per endpoint was necessary to show that we could layout the network in area linear in the number of compute blocks. However, it is not sufficient to show that we can use additional wire layers to achieve a compact layout. For unconstrained logic, it is not clear that more wire layers will always be usable. For example, [13] argues that wiring on an upper layer metal plane will occupy 12–15% of all the layers below it. Integrating this result across wire planes, this argues a useful limit of 6–7 wiring levels. The MoT wiring topology, however, is quite stylized with geometrically increasing wire lengths. Consequently, it does not exhibit the saturation effect which we would get with unconstrained netlists. Rather than consuming a constant fraction of lower layers, as [13] assumes, each additional metal layer in the MoT layout uses up a fraction of the layers below it that decreases geometrically with the layer height. In fact, we can show that a design which needs $O(f(N)) > O(\sqrt{N})$ bisection bandwidth can be laid out with only $O(\max(f(N)/\sqrt{N}, 1))$ wiring layers. More specifically, for any $p > 0.5$, we can use $O(N^{(p-0.5)})$ layers and maintain a constant channel width.

1) *Binary Tree* ($p = 0.5$): To build intuition, let us focus initially on the binary tree case ($p = 0.5$). The key observation is that we can layout each binary tree along its row (or column) using $O(\log(l_{\text{row}}))$ wiring layers in a strip which is $O(1)$ wide and runs the length of the row ($l_{\text{row}} = \sqrt{N}$).

Fig. 5 shows how the row (column) tree is mapped into a 1-D layout with $O(\log(N))$ wiring layers. It is important to notice that each subtree layout leaves one free switch location for an upper level switch. When we combine two subtrees, we can place the switch connecting them in one of the two free slots, leaving a single slot free in the resulting subtree. In this manner, the recursive composition of subtrees can continue indefinitely; the geometrically increasing via spacing allows it to avoid ever running out of via area on the lower levels of metallization. As shown, each new tree level simply adds one additional wire run above the existing wires. This $p = 0.5$ case requires $\Theta(\log(N))$

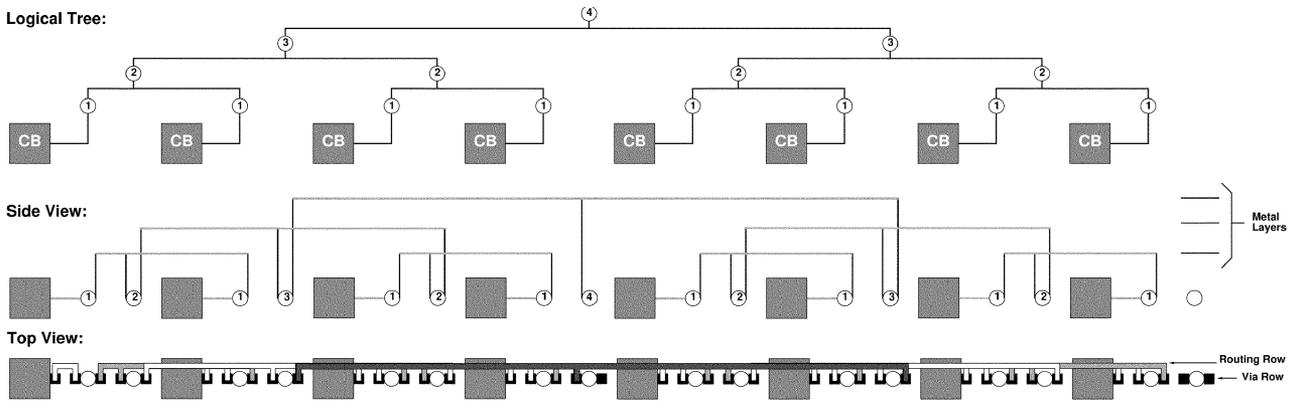


Fig. 5. One-dimensional binary tree layout.

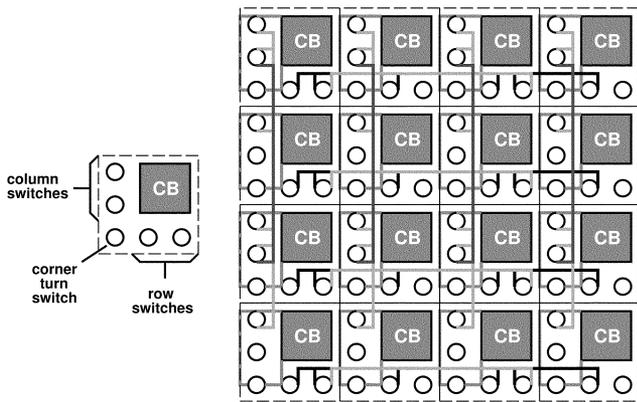


Fig. 6. Minimal MoT layout.

metal layers, which is asymptotically optimal to accommodate the $\log(N)$ wires which each tree contributes to each row or column. Note that if we make the width of the row as wide as a via and a wire, we can bring all the wires up to the appropriate metal layer without interfering with the row wire runs (See the “Top View” in Fig. 5).

In practice, the width of a switch is likely to be several wire pitches wide; consequently, we can place several tree levels in a single metal layer and run them within the width of the switch row; this means that the number of wire layers we need for each row (or column) layout in practice is $\log_2(\sqrt{N})/r$ where r is the ratio of the switch width to wire pitch (strictly speaking one less than that to accommodate the via row). For example, if the switch width is 50λ and the wire pitch is 8λ , we can put six wires within the width of the switch. If we use one track for vias, this means we can place five tree levels on each wire layer, so the number of layers needed to accommodate the row (column) tree is $\lceil \log_2(\sqrt{N})/5 \rceil$.

The full MoT structures requires both row and column trees. We must space out the row and column switches to accommodate the cross switches. Further, we must assign separate wire layers for the rows and columns. Together, this means we will need $\lceil 2 \cdot \log_2(\sqrt{N})/r \rceil$ layers for wiring. In practice, additional wiring layers will be needed for power, ground, and clock routing.

Fig. 6 shows a minimal layout with a single tree in each row and column channel. In practice, we will typically use several trees ($C > 1$) in each row and column and require C-box

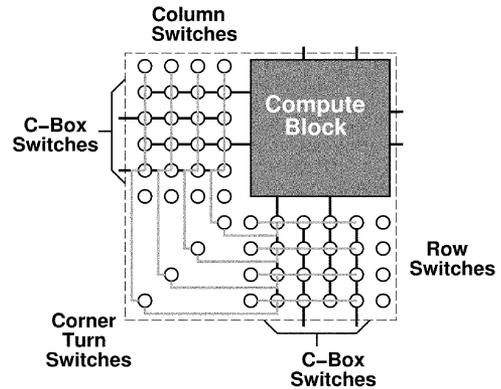


Fig. 7. MoT tile with $C = 4$ and $I = 2$.

switches. Fig. 7 shows the base tile for a larger network configuration.

2) *Fatter Trees* ($p > 0.5$): This same basic layout scheme works for the case where $0.5 \leq p < 1.0$. We will not always have exactly half as many switches on each immediately successive tree level. However, as long as $p < 1.0$, there are a number of tree stages over which the number of switches will be half the number of switches in the preceding group of tree stages. By grouping the switches into these groups, we can use the same strategy shown for the binary tree case.

Fig. 4 shows the switch arrangement for the aforementioned $p = 0.75$ case. It should be clear from the layout tree diagrams that the switches can be shuffled to the base layer as in Fig. 5. Here, we will, asymptotically, end up with six switches between every pair of compute blocks (25). Up to a span of 16 endpoints, we need five switches (Table II). Beyond that, each pair of stages contributes half as many switches as the previous pair of stage, resulting in a total of one more switch per endpoint. As we compose each additional pair of stages we end up leaving half of the remaining slots in each span with space for switches from the next span. As shown in Fig. 4, we spread the uplinks across the entire width of the stage. This filling can continue indefinitely just as the $p = 0.5$ filling we have already seen.

Notice that the total number of metal layers is asymptotically optimal. That is, for $p > 0.5$, we must have $O(N^{p-0.5})$ wires in the top level of the channel to accommodate bisection requirements. To make the channels $O(1)$ wide, we must use $O(N^{p-0.5})$ layers to accommodate the bisection require-

ments. The channel actually needs to accommodate all the wire levels of the MoT. Since there are geometrically fewer wires in each lower level of each row or channel MoT tree when $p > 0.5$, when we sum the total number of wires across all levels in each row or column tree, the total wire count is simply a constant factor times the number of wires in the top channel; this is developed further in Section V-A3. Consequently, the total number of wire layers is $O(N^{(p-0.5)})$.

D. Total Hierarchical Wires

As we saw in Section II-C, when we have hierarchical wiring, the total numbers of wires in the channel, W_x , depends on the level. For the MoT, the level is defined by the size of the device (21). The total number of wires in a channel is then

$$W_x = C \left(\sum_{i=0}^l w_{\text{ch}}(i) \right). \quad (27)$$

Equation (22) told us how to calculate $w_{\text{ch}}(i)$ for the $p = 0.75$ case. Fig. 4 shows a MoT channel with $C = 1$, $p = 0.75$; $w_{\text{ch}}(4) = 4$ and $W_x = 13$. Later (30), we will see how to calculate $w_{\text{ch}}(i)$ for any p .

E. Switch Dominated

The asymptotic reduction in switching requirements compared to the Manhattan topology makes wiring requirements more likely to be a limiting factor. At the same time, however, this topology allows us to maximally use additional metal layers. As a consequence, the MoT designs will always be switch area dominated when given sufficient layers of interconnect.

F. Delay

Note that switch delay is asymptotically logarithmic in the distance between the source and the destination. A route simply needs to climb the tree to the appropriate tree level to link to the destination row or column, then descend the tree. It is also worthwhile to note that the stub capacitance associated with each level of the tree is constant. That is, there are a constant number of switches (drivers or receivers) attached to each wire segment, regardless of its length. This is an important contrast with the flat, Manhattan connection scheme where the number of switches attached to a long wire is proportional to the length of the wire. An added benefit of the strict hierarchy is that it manages to minimize the switch capacitance associated with long wire runs.

Buffered switches are needed to achieve minimum delay and to isolate each wire segment from the fanout that may occur on a multipoint net.

G. Long Wire Runs

Ultimately, we will need to buffer the long wire runs in order to achieve linear delay with interconnect length and minimize the delay traveling long distances. This will end up forcing us to insert buffers at fixed distances which can reduce the benefits of the convenient geometric switching property identified. Technological advances that provided linear delay with distance

without requiring repeaters (*e.g.*, optical, superconducting wires) would obviate this problem.

H. Relation to Tree-of-Meshes

Agarwal [14], Lai [15], and Tsu [16] have previously described hierarchical FPGA interconnect architectures. DeHon showed that the butterfly fat-tree style interconnect of the hierarchical synchronous reconfigurable array (HSRA) could also be laid out in constant area given sufficient wire layers for the $p = 0.5$ case [5]. These networks all build a single, unified hierarchy and are closely related to the tree-of-meshes topology [7]. In contrast, the MoT used here is directly a 2-D structure building hierarchical routing along each row and column. As such, the MoT can be viewed as a hybrid between the strict, single hierarchy of the tree-of-meshes and the nonhierarchical Manhattan array. A rigorous comparison of the tree-of-meshes and the MoT is addressed in [2121].

IV. EMPIRICAL EXPERIMENTS

In Section III, we demonstrated the favorable asymptotic switching requirements for the MoT design assuming we can contain the number of required base channels, C , to a suitably small constant. In this section we show empirically that the base channel requirements are uniformly small. Further, we show that even for the small sizes of conventional FPGA benchmarks, the MoT scheme already shows some practical advantages in reducing aggregate switch requirements.

A. Base Comparison

For a base level comparison, we use the benchmarks from Toronto's "FPGA place and route challenge" [17] to compare the channel, domain, and switch requirements between the traditional Manhattan routing topology and our MoT topology. We used the `vpr422_challenge_arch` architecture as the baseline mesh; this has single-length segments and a single LUT per Island. We substitute a universal switch [18] for the subset switch used in the `vpr422` challenge because the routed mesh designs using universal switches uniformly require less switches than the subset-switch-based designs. Each of the 4 LUT inputs appears on a single side of the logic block ($T_{\text{in}} = 1$), and the output appears on two sides ($T_{\text{out}} = 2$); both are fully populated ($F_c = 1$) [see Fig. 8(a)]. We use VPR 4.3 to produce the placed designs for **both** the Manhattan and MoT routing. We use the channel minimizing VPR 4.3 router to route the Manhattan designs. Since prior work suggested the superiority of longer segments [9], [12], we also routed a uniform, length-4 segment Manhattan case for comparison; all other parameters are identical to the base length-1 Manhattan case.

For our overall comparison, we assembled a MoT design with $T = 1$ [see Fig. 8(b)]. We developed our own, Pathfinder-based [19] router to route the MoT designs. To match the VPR-style results, we let the number of base channels, C , float and report the minimum number of channels required to route the design for various p values.

Table III summarizes these basic results. For almost all designs, the MoT routes with sufficiently small C as to require fewer total switches than the Manhattan designs. Overall, the

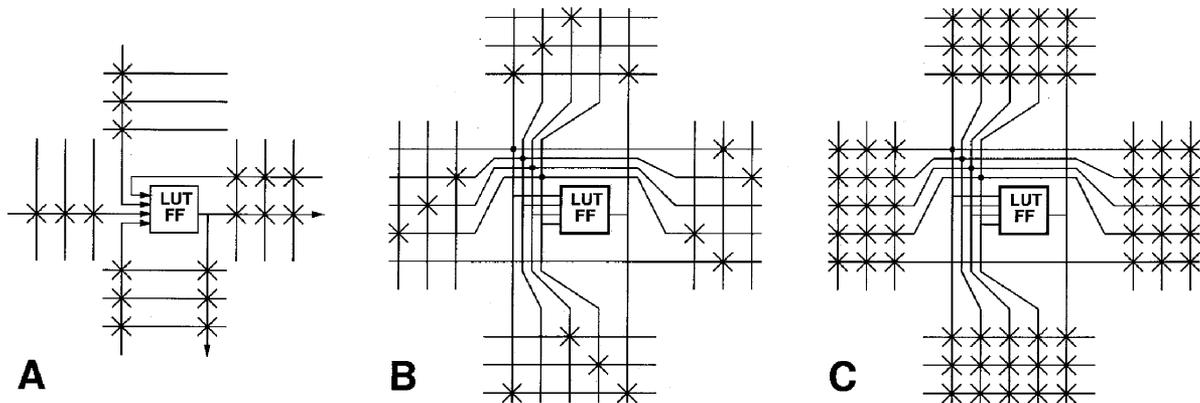

 Fig. 8. Logic block IO structure (all shown $W = 3$).

 TABLE III
 MANHATTAN VERSUS MESH-OF-TREES

Design	#LB	Manhattan (universal)				Mesh of Trees			
		$L_{seg} = 1$		$L_{seg} = 4$		$p = 0.67$		$p = 0.75$	
Circuit		W	B_{sw}	W	B_{sw}	C	B_{sw}	C	B_{sw}
alu4	1522	9	109	13	110	4	86	4	101
apex2	1878	10	121	14	118	5	106	4	98
apex4	1262	11	134	15	127	5	110	5	129
bigkey	1707	6	73	10	84	3	62	3	72
clma	8382	10	121	14	117	5	103	4	96
des	1591	7	85	10	84	3	65	3	79
diffeq	1497	6	73	9	76	4	88	3	77
dsip	1370	6	73	10	84	3	62	3	72
elliptic	3604	9	109	12	101	4	82	4	93
ex1010	4598	10	121	13	109	5	107	4	102
ex5p	1064	12	146	15	127	5	113	4	106
frisc	3556	11	133	14	117	5	103	4	94
misex3	1397	10	122	13	110	5	108	4	100
pdv	4575	14	169	18	150	6	128	5	128
s298	1931	6	73	10	84	4	84	3	73
s38417	6406	7	85	10	84	4	84	3	75
s38584.1	6446	7	85	10	84	4	84	4	100
seq	1750	10	122	13	109	5	107	4	98
spla	3690	12	145	16	134	6	123	5	117
tseng	1047	6	73	9	76	4	90	3	80
sum		179	2172	248	2085	89	1895	76	1890

simple MoT requires 9% fewer switches than the mesh networks.

While the MoT designs require fewer base channels (C) than the Manhattan designs require wire channels (W), we noted in Section III-D that the total wires across all levels (W_x) will be larger. For `clma` for instance, $W = 14$ for the segment length 4 Manhattan design, while $C = 5$ for the $p = 0.67$ MoT. The $C = 5$ MoT will have $W_x = 135$. As we will see in Section V, the MoT wiring can be reduced using higher arity trees.

B. Small C 's

The C 's are uniformly small, many being as low as 3 for $p = 0.75$. We will see many more reduced to 3 or 4 with the variations in Section V. The C required for the design is driven by three things:

- 1) bisection bandwidth;
- 2) number of distinct signals which must enter a channel;
- 3) domain coloring limitations.

A sufficiently large p value can generally accommodate bisection needs (see Fig. 4). For channel entrance, note that a fully

used k -LUT with a single output needs to have $k + 1$ potentially distinct signals enter one of the four channels which surrounds it. Further note that it shares each of those channels with another k -LUT which has similar requirements. Consequently, the channel entrance lower bound is

$$C_{lb} \geq \left\lceil \frac{2 \cdot (k + 1)}{4} \right\rceil. \quad (28)$$

For $k = 4$, $C_{lb} = 3$. Finally, since the MoT design described here maintains the domain topology typical of Manhattan FPGA interconnect, it could have colorability limitations [20]. The routed results suggest that the colorability issues are not a major issue in practice as we are close to the channel entrance lower bound on all designs.

V. MOT VARIATIONS

There are many options for detailed construction of MoT networks which retain the good asymptotic characteristics derived in Sections II–IV while, perhaps, offering lower absolute switching and wiring requirements. In this section, we review a few options including arity, shortcuts, and staggering.

A. Arity

Arity refers to the branching factor as we go down the tree—the number of children segments associated with each parent segment in the tree. Fig. 9 shows arity-2 and arity-4 trees.

Higher-arity makes the trees flatter. This will reduce the number of switches in a path, but will increase the capacitance associated with each segment along the path. Higher arity reduces the wires per domain, but can increase the number of domains needed to route a design. In particular, higher arity can force a number of short connections which would have been disjoint to now overlap [see Fig. 10(b)].

In the extreme, we completely flatten each channel into an arity- \sqrt{N} tree. This would be equivalent to building a crossbar along each column of tree. Such a crossbar would have channel width \sqrt{N} , worse than the mesh, forcing a total number of switches which grow as $N^{1.5}$. So, clearly, we can make the arity too large to exploit the structure of designs. At the other extreme, arity-2 designs may have too many switches in the path and more intermediate levels than are useful. The challenge is

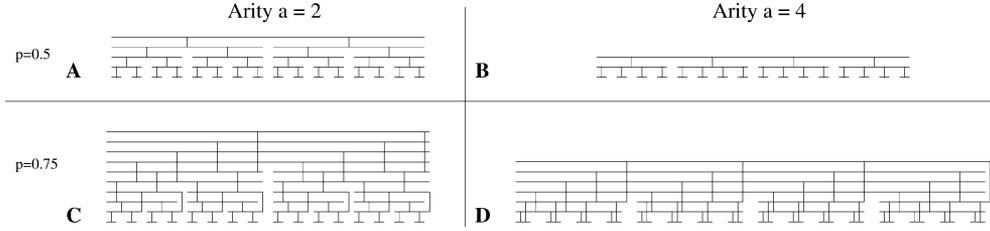


Fig. 9. Single channel in 16 node row or column tree shown with various arities and rent growth rates.

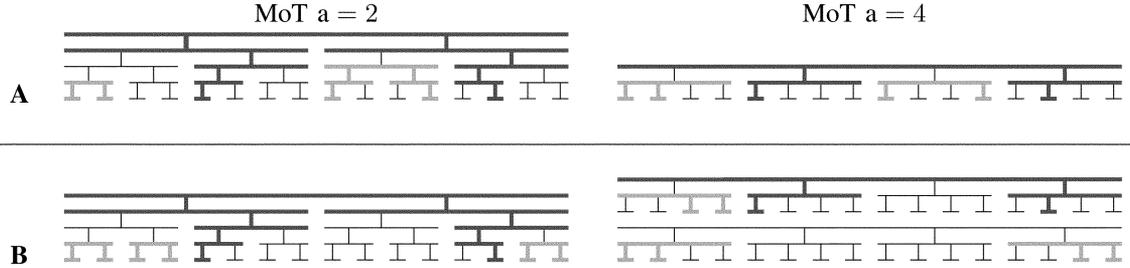


Fig. 10. MoT wiring versus arity.

to find the best balance point between these extremes which allows us to maximally exploit design structure.

1) *Arity, Growth Rate, and Rent Exponent:* For any arity, we can approximate any Rent Exponent growth rate by selecting the appropriate sequence of channel growths. Let $a =$ arity, $l =$ levels, $g_i =$ growth at level i . A MoT with l levels has a total number of nodes, N

$$N = (a^l)^2 = a^{2l}. \quad (29)$$

The width of the top channel of a tree is

$$w_{\text{ch}}(l) = \prod_{i=0}^l g_i. \quad (30)$$

The total bisection width, BW, of a level l MoT is

$$\text{BW}(l) = C \cdot w_{\text{ch}}(l) \cdot \sqrt{N} = C \cdot w_{\text{ch}}(l) \cdot a^l. \quad (31)$$

Using our Rent Relation (7), we have

$$\text{BW}(l) = c_r N^p = C \cdot w_{\text{ch}}(l) \cdot a^l \quad (32)$$

$$c_r (a^{2l})^p = C \left(\prod_{i=0}^l g_i \right) \cdot a^l. \quad (33)$$

Here we see directly that the tree domain C plays a similar role to the Rent constant multiplier, c_r . To look at the growth effects, we drop these constants, understanding that we can use them to provide a constant shift on the bandwidth curves

$$a^{2lp} = \left(\prod_{i=0}^l g_i \right) \cdot a^l \quad (34)$$

$$2lp \log a = \log \left(\prod_{i=0}^l g_i \right) + l \log a \quad (35)$$

$$p = \frac{\log \left(\prod_{i=0}^l g_i \right)}{2l \log a} + \frac{1}{2}. \quad (36)$$

For $p = 0.5$, we simply need $g_i = 1$, for all i . For $p = 0.75$, we pick the g_i sequence to correspond to the arity. For $a = 2$, this means we make even g_i 's be two and odd g_i 's be one as previously noted in Section III-B [see Fig. 9(c)], so

$$w_{\text{ch}}(l, p = 0.75) = \left(\prod_{i=0}^l g_i \right) = 2^{(l/2)}. \quad (37)$$

Putting that in (36)

$$p = \frac{\log [2^{(l/2)}]}{2l \log 2} + \frac{1}{2} = \frac{3}{4}. \quad (38)$$

For $a = 4$, we make all g_i 's be 2 [see Fig. 9(d)]

$$p = \frac{\log [2^{(l)}]}{2l \log 4} + \frac{1}{2} = \frac{3}{4}. \quad (39)$$

Table IV summarizes the growth sequences used in this work.

2) *Switches Per Domain:* Amortized across the \sqrt{N} endpoints sharing a single horizontal or vertical domain tree, each endpoint will have a number of tree switches

$$T_{\text{sw}} = g_0 + \frac{g_0 \cdot g_1}{a} + \frac{g_0 \cdot g_1 \cdot g_2}{a^2} + \frac{g_0 \cdot g_1 \cdot g_2 \cdot g_3}{a^3} + \dots \quad (40)$$

Equation (40) is the generalization of the arity-2, p -specific switch counting illustrated in (18) and (25). Assuming the g_i 's are powers of two, Fig. 11 shows how the number of switches per domain varies with the arity. For $p = 0.5$, higher arities have fewer stages and hence less switches per endpoint, as should be clear from Fig. 9. For $p > 0.5$, there are two competing effects. Higher arities have fewer stages, but the higher arity results in flattening that forces each uplink to connect to a greater number

TABLE IV
GROWTH SEQUENCES USED TO ACHIEVE VARIOUS ARITY, RENT EXPONENT
DESIGNS IN THIS ARTICLE

Arity	p	Growth Sequence
2	0.67	(2 1 1)*
2	0.75	(2 1)*
3	0.65	(2 1)*
3	0.81	(2)*
4	0.625	(2 1)*
4	0.67	(2 2 1)*
4	0.75	(2)*
5	0.60	(2 1)*
8	0.67	(2)*

of parents. As shown, this results in a minimum number of switches per domain around an arity of 4 for $p = 0.75$. The odd powers of two end up being less efficient than the even powers in our discrete approximation to $p = 0.75$, causing the nonmonotonic growth in Fig. 11. Fig. 11 is not the full story as we must also understand how the number of tree domains changes (Section V-A.4) as we go to larger arities.

3) *Wires Per Domain*: In general, the number of wires per domain, w_x , is the sum of the channels required at each level of each base tree ($w_{ch}(l)$, where l is the tree level)

$$\begin{aligned}
 w_x(l) &= w_{ch}(l) + w_{ch}(l-1) + w_{ch}(l-2) + \dots \\
 &= \left(\prod_{i=0}^l g_i \right) + \left(\prod_{i=0}^{l-1} g_i \right) + \left(\prod_{i=0}^{l-2} g_i \right) + \dots \\
 &= \left(\prod_{i=0}^l g_i \right) \left(1 + \frac{1}{g_l} + \frac{1}{g_l \cdot g_{l-1}} + \dots \right). \quad (41)
 \end{aligned}$$

For $p > 0.5$, it will always be the case that there are g_i 's greater than one such that this $1/g_i$ series summation converges to a constant as l approaches infinity. For arity 2 and $p = 0.75$, the series converges to 3.5, while for arity 4 and $p = 0.75$, it converges to 2; Fig. 9 shows this effect graphically. For all p 's, higher arity implies higher growth rates and fewer terms in the sum resulting in fewer total wire channels.

4) *Domains*: An important question that the prior sections raise is: how do the number of domains (C) change with arity? As Fig. 10 shows, sometimes a higher arity MoT will require no more base channels than the arity-2 MoT, resulting in a net decrease in total wires. Other times, the MoT may require a factor of $a/2$ more base channels than the arity-2 MoT.

5) *Empirical Results*: Tables V and VI summarize the resources required when each of the Toronto 20 benchmarks is routed to minimize the number of tree domains (C) required to route the design. Note from Table VI that the C 's do not increase with arity near the worst-case ratios; in fact there is little growth for the cases shown. In Table V, we see an arity-4, $p = (2/3)$ MoT requires only 1634 switches, about 14% fewer than the switches required by an arity-2 MoT.

The total number of wires in the channel, W_x , (see Section III-D) is the product of the number of domains, C , and the wires per domain, w_x (41). Table VI further shows us that the arity-5, $p = 0.6$ MoT requires only $1107/1907 \approx 58\%$ of the wires required by the smallest arity-2 MoT. The total MoT channel wires (W_x) are still larger than the mesh channel widths

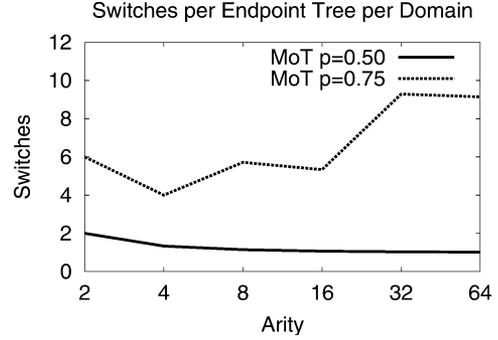


Fig. 11. Switches per endpoint per tree per domain.

TABLE V
TOTAL SWITCHES VERSUS ARITY AND RENT EXPONENT (p)

arity	2		3		4			5	8
	0.67	0.75	0.65	0.81	0.625	0.67	0.75	0.60	0.67
alu4	86	101	88	94	95	88	74	91	90
apex2	106	98	88	95	109	87	91	105	103
apex4	110	129	108	99	113	89	95	123	104
bigkey	62	72	52	71	62	51	54	60	58
clma	103	96	86	99	106	85	93	100	103
des	65	79	69	69	63	71	60	58	61
diffreq	88	77	71	72	64	71	75	61	76
dsip	62	72	70	71	62	68	54	60	58
elliptic	82	93	85	91	91	84	71	88	86
ex1010	107	102	84	90	93	88	98	101	106
ex5p	113	106	108	99	114	90	96	123	107
frisc	103	94	85	91	91	85	89	103	101
misex3	108	100	89	97	95	87	93	106	88
pdc	128	128	118	112	124	124	117	144	136
s298	84	73	70	71	62	69	72	74	73
s38417	84	75	70	77	61	69	76	72	74
s38584.1	84	100	70	77	77	69	76	72	74
seq	107	98	104	93	93	86	91	105	104
spla	123	117	101	91	106	101	106	117	114
tseng	90	80	72	74	65	72	77	62	76
max	128	129	118	112	124	124	117	144	136
sum	1895	1890	1688	1733	1746	1634	1658	1825	1792

(W). The worst-case W_x for the arity = 5, $p = 0.6$ MoT is 90, while the worst-case W for the segment length 4 Mesh is 18. The 90 total MoT wires divide into nine wires used by each of ten domains. Conservatively assuming a minimum size $50\lambda \times 50\lambda$ switch and an 8λ wire pitch, we can route the nine wires in each domain over the width of a single switch in two metal layers since we get $r = 6$ wire pitches per layer (see Figs. 5 and 7). Two wire layers for horizontal channel routing plus two layers for vertical channel routing mean we will need only four routing layers to layout this design so that it is the switch area which determines device density not the wiring. Consequently, the fact that the MoT reduces switches at the expense of wires compared to the mesh results in a net decrease in device area.

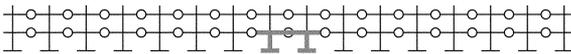
B. Shortcuts and Staggering

Perhaps the biggest concern about the MoT relative to a pure Manhattan design is the fact that some nodes which are physically close in the layout will not be logically close in the tree. This can, in the worst case, require a wire to travel through $2 \log_a(\sqrt{N})$ switches when two would suffice in the mesh or

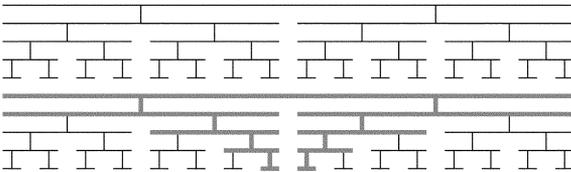
TABLE VI
TREE DOMAINS (C) AND WIRES PER CHANNEL (W_x) VERSUS ARITY AND RENT EXPONENT (p)

arity	2				3				4				5		8			
	0.67		0.75		0.65		0.81		0.625		0.67		0.75		0.60		0.67	
p	C	W_x	C	W_x	C	W_x	C	W_x	C	W_x	C	W_x	C	W_x	C	W_x	C	W_x
alu4	4	76	4	116	5	65	4	124	6	54	5	55	4	60	6	54	6	42
apex2	5	95	4	116	5	65	4	124	7	63	5	55	5	75	7	63	7	49
apex4	5	95	5	145	6	78	4	124	7	63	5	55	5	75	8	72	7	49
bigkey	3	57	3	87	3	39	3	93	4	36	3	33	3	45	4	36	4	28
clma	5	135	4	180	5	105	4	252	7	91	5	95	5	155	7	63	7	105
des	3	81	3	135	4	52	3	93	4	52	4	76	3	93	4	36	4	60
diffeq	4	76	3	87	4	52	3	93	4	36	4	44	4	60	4	36	5	35
dsip	3	57	3	87	4	52	3	93	4	36	4	44	3	45	4	36	4	28
elliptic	4	76	4	116	5	65	4	124	6	54	5	55	4	60	6	54	6	42
ex1010	5	135	4	180	5	65	4	124	6	78	5	95	5	155	7	63	7	105
ex5p	5	95	4	116	6	78	4	124	7	63	5	55	5	75	8	72	7	49
frisc	5	95	4	116	5	65	4	124	6	54	5	55	5	75	7	63	7	49
misex3	5	95	4	116	5	65	4	124	6	54	5	55	5	75	7	63	6	42
pdc	6	162	5	225	7	91	5	155	8	104	7	133	6	186	10	90	9	135
s298	4	76	3	87	4	52	3	93	4	36	4	44	4	60	5	45	5	35
s38417	4	108	3	135	4	84	3	189	4	52	4	76	4	124	5	45	5	75
s38584.1	4	108	4	180	4	84	3	189	5	65	4	76	4	124	5	45	5	75
seq	5	95	4	116	6	78	4	124	6	54	5	55	5	75	7	63	7	49
spla	6	114	5	145	6	78	4	124	7	63	6	66	6	90	8	72	8	56
tseng	4	76	3	87	4	52	3	93	4	36	4	44	4	60	4	36	5	35
max	6	162	5	225	7	105	5	252	8	104	7	133	6	186	10	90	9	135
sum	89	1907	76	2572	97	1365	73	2583	112	1144	94	1266	89	1767	123	1107	121	1143

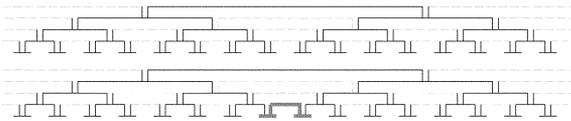
A: Manhattan



B: MoT



C: MoT w/ shortcuts



D: MoT w/ domain staggering

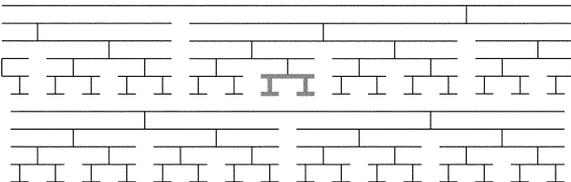


Fig. 12. Mitigating tree discontinuities with shortcuts and staggering.

MoT had this connection aligned differently with respect to the tree [see Fig. 12(a), (b), and (d)].

1) *Shortcuts*: Shortcut connections which bridge these hierarchy gaps [see Fig. 12(c)] can guarantee that a signal never needs to be routed higher in the tree than $O(\log_a(\text{Manhattan Distance}))$. This also guarantees the

total distance traversed is within a constant factor of the Manhattan Distance between the source and sink.

Full shortcut population can double the tree switches per domain required for an arity-2 MoT (T_{sw}). The general increase is

$$T_{sw\text{-short}} = \left(\frac{a+2}{a} \right) T_{sw}. \quad (42)$$

That is, we add one switch at each end of the arity group in addition to the a switches already providing up tree connections.² Table VII shows that while the shortcuts do reduce the number of domains (C) and the total wiring (W_x), their inclusion results in a net increase in switch count.

2) *Staggering*: Most of the worst tree alignment effects can be reduced simply by staggering the domains relative to each other. This way, if there is a bad break on one tree, there will be a more favorable sibling relationship on another tree [see Fig. 12(d)]. In Section IV-B we established the minimum number of tree domains needed simply to exit all five IOs for a 4-LUT is $C = 3$, so there will always be multiple domains available for staggering. With larger clusters, the minimum domain size increase. Staggering the domains requires no additional switches and will often reduce the number of domains needed to route a design.

Table VIII summarizes the mapped resource requirements when we stagger the tree domains in the channel. This shows an additional reduction of 14% in total wiring requirements (W_x) and 5% in total switches (B_{sw}). Overall, this brings the switch saving compared to the mesh to 26%.

²This can actually be $(a+1)/a$ as we only need one switch to enable or disable the shortcut so we only need to charge half a switch to each of the two segments being connected. However, as we get higher in the tree, having a single switch could result in a long wire stub. In practice, we might use one switch at the lower levels and two at the upper levels.

TABLE VII
PROPERTY SUMMARY VERSUS ARITY FOR $p = 0.67$ WITH SHORTCUTS

property	arity-2 $p=0.67$			arity-4 $p=0.67$			arity-8 $p=0.67$		
	W_x	C	B_{sw}	W_x	C	B_{sw}	W_x	C	B_{sw}
alu4	76	4	138	44	4	99	35	5	102
apex2	76	4	135	55	5	122	42	6	119
apex4	76	4	139	55	5	126	42	6	121
bigkey	57	3	100	33	3	72	28	4	78
clma	108	4	133	76	4	97	90	6	119
des	81	3	106	57	3	76	60	4	83
diffeq	57	3	106	44	4	100	28	4	82
dsip	57	3	100	33	3	72	28	4	78
elliptic	76	4	132	44	4	95	35	5	98
ex1010	108	4	137	76	4	100	75	5	102
ex5p	76	4	143	55	5	127	42	6	124
frisc	76	4	132	44	4	96	35	5	98
misex3	76	4	137	44	4	99	35	5	100
pdic	108	4	137	95	5	125	105	7	143
s298	57	3	101	44	4	97	28	4	79
s38417	81	3	102	76	4	99	60	4	81
s38584.1	81	3	102	76	4	99	60	4	81
seq	76	4	136	55	5	122	42	6	121
spla	76	4	132	55	5	119	42	6	117
tseng	57	3	108	33	3	76	28	4	83
max	108	4	143	95	5	127	105	7	143
summary	1536	72	2456	1094	82	2018	940	100	2009

TABLE VIII
PROPERTY SUMMARY VERSUS ARITY AND RENT EXPONENT (p) WITH DOMAIN STAGGERING

property	arity p	2	3	4		5	8
		0.67	0.65	0.625	0.67	0.6	0.67
W_x	max	135	105	91	114	72	120
	sum	1712	1287	1042	1184	954	1085
C	max	5	6	7	6	8	8
	sum	80	91	102	88	106	115
B_{sw}	max	116	105	110	107	118	122
	sum	1739	1628	1610	1553	1606	1717

VI. SUMMARY AND FUTURE WORK

Using the MoT topology, we can achieve better scalability than a flat, Manhattan topology. Assuming the number of base channels, C , remains constant for increasing design size, the total number of switches per LUT in our MoT converges to a constant $[\Theta(1)]$ independent of design size; this should be contrasted with the $\Omega(N^{p-0.5})$ switches per LUT required for a flat, Manhattan topology. Given sufficient wiring layers, the MoT network layout can maintain a constant area per logic block as the design scales up. Asymptotically, the number of switches in any path in the MoT needs to only grow as $O(\log(N))$. Our initial empirical experiments verify small C values that show no signs of growing with design size and total switch requirements that are 26% smaller than those of conventional mesh designs. In the process we show that arity-4 trees require the least absolute switches with less than 70% of the wiring requirements of arity-2 trees.

In this paper, we have focused on the resource requirements for the MoT, but have not treated the absolute delay or energy

implications. A careful accounting and comparison of delay and energy are important pieces of future work which will be necessary to establish the practical viability of this scheme. We have explored many of the parameters associated with designing MoT networks but additional design parameters deserve study including Island-style clustering, flattening, corner turns, and depopulated shortcuts. We expect larger benchmarks will better demonstrate the scalability of this architecture.

REFERENCES

- [1] W. S. Carter, K. Duong, R. H. Freeman, H.-C. Hsieh, J. Y. Ja, J. E. Mahoney, L. T. Ngo, and S. L. Sze, "A user programmable reconfigurable logic array," in *IEEE 1986 Custom Integrated Circuits Conf.*, May 1986, pp. 233–235.
- [2] M. Bohr, "Interconnect scaling—The real limiter to high performance ulsi," in *Int. Electron Devices Meeting 1995 Tech. Dig.*, Dec. 1995, pp. 241–244.
- [3] B. S. Landman and R. L. Russo, "On pin versus block relationship for partitions of logic circuits," *IEEE Trans. Computers*, vol. 20, pp. 1469–1479, 1971.
- [4] A. DeHon, "Rent's rule based switching requirements," in *Proc. System-Level Interconnect Prediction Workshop*, Mar. 2001, pp. 197–204.
- [5] —, "Compact, multilayer layout for butterfly fat-tree," in *Proc. 12th ACM Symp. Parallel Algorithms Architectures*, July 2000, pp. 206–215.
- [6] F. T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*: Morgan Kaufmann, 1992.
- [7] —, "New lower bound techniques for VLSI," in *Proc. Annual Symp. Foundations of Computer Science*, 1981.
- [8] S. D. Brown, R. J. Francis, J. Rose, and Z. G. Vranesic, *Field-Programmable Gate Arrays*. Norwell, MA: Kluwer, 1992.
- [9] V. Betz, J. Rose, and A. Marquardt, *Architecture and CAD for Deep-Submicron FPGAs*. Norwell, MA: Kluwer, 1999.
- [10] W. E. Donath, "Placement and average interconnection lengths of computer logic," *IEEE Trans. Circuits Syst.*, vol. 26, pp. 272–277, Apr. 1979.
- [11] A. E. Gamal, "Two-dimensional stochastic model for interconnections in master slice integrated circuits," *IEEE Trans. Circuits Syst.*, vol. 28, pp. 127–138, Feb. 1981.
- [12] V. Betz and J. Rose, "FPGA routing architecture: Segmentation and buffering to optimize speed and density," in *Proc. Int. Symp. Field-Programmable Gate Arrays*, Feb. 1999, pp. 59–68.
- [13] G. A. Sai-Halasz, "Performance trends in high-end processors," *Proc. IEEE*, vol. 83, pp. 20–36, Jan. 1995.
- [14] A. A. Agarwal and D. Lewis, "Routing architectures for hierarchical field programmable gate arrays," in *Proc. 1994 IEEE Int. Conf. Computer Design*, Oct. 1994, pp. 475–478.
- [15] Y.-T. Lai and P.-T. Wang, "Hierarchical interconnection structures for field programmable gate arrays," *IEEE Trans. VLSI Syst.*, vol. 5, pp. 186–196, June 1997.
- [16] W. Tsu, K. Macy, A. Joshi, R. Huang, N. Walker, T. Tung, O. Rowhani, V. George, J. Wawrzynek, and A. DeHon, "HSRA: High-speed, hierarchical synchronous reconfigurable array," in *Proc. Int. Symp. Field-Programmable Gate Arrays*, Feb. 1999, pp. 125–134.
- [17] V. Betz and J. Rose. (1999) FPGA Place-and-Route Challenge. [Online]. Available: <http://www.eecg.toronto.edu/~vaughn/challenge/challenge.html>
- [18] Y.-W. Chang, D. F. Wong, and C. K. Wong, "Universal switch-module design for symmetric-array-based FPGA's," in *Proc. Int. Symp. Field-Programmable Gate Arrays*, Feb. 1996, pp. 80–86.
- [19] L. McMurichie and C. Ebling, "Pathfinder: A negotiation-based performance-driven router for FPGAs," in *Proceedings of the International Symposium on Field-Programmable Gate Arrays*, Feb. 1995, pp. 111–117.
- [20] Y.-L. Wu, S. Tsukiyama, and M. Marek-Sadowska, "Graph based analysis of 2-d FPGA routing," *IEEE Trans. Computer-Aided Design*, vol. 15, pp. 33–44, Jan. 1996.
- [21] A. DeHon, "Unifying mesh- and tree-based programmable interconnect," *IEEE Trans. TVLSI Syst.*, vol. 12, pp. 1051–1065, Oct. 2004.



André DeHon (S'92–M'96) received the S.B., S.M., and Ph.D. degrees in electrical engineering and computer science from the Massachusetts Institute of Technology, Cambridge, in 1990, 1993, and 1996, respectively.

From 1996 to 1999, he was Co-Head of the BRASS Group, Computer Science Department, University of California, Berkeley. Since 1999, he has been an Assistant Professor of Computer Science at the California Institute of Technology. He is broadly interested in how we physically implement

computations from substrates, including VLSI and molecular electronics, up through architecture, CAD, and programming models. He places special emphasis on spatial programmable architectures (e.g., field-programmable gate-arrays) and interconnect design and optimization.



Raphael Rubin received the B.S. degree in engineering and applied sciences from the California Institute of Technology, Pasadena, CA, in 2001.

Since 2001, he has worked as a Research Associate for the Implementation of Computation Group in the Computer Science Department, California Institute of Technology. His research interests include programmable interconnect and design automation.