CHAPTER 6

Cluster-Based Logic Blocks

In this chapter we investigate the speed and area-efficiency of FPGAs which use *logic clusters* as their logic block. A logic cluster is composed of several look-up tables and registers interconnected by local routing, as described in Section 3.1.1. In the next section we motivate our research by describing some of the advantages of cluster-based logic blocks, and by showing that these logic blocks are commercially relevant. Section 6.2 describes the experimental flow we use to evaluate different logic clusters. Sections 6.3 through 6.6 then explore several key architectural questions concerning these logic blocks: how many inputs (I) should the FPGA routing provide to each logic cluster; how should the logic block to general routing interface change as a function of logic cluster size (N); and how are circuit speed, FPGA area-efficiency, and design compile time affected by the size of the logic cluster used?

6.1 Motivation

As described in Section 2.1.2, most SRAM-based FPGAs use logic blocks based on look-up tables (LUTs). A look-up table with more inputs can implement more logic, and hence one needs fewer logic blocks to implement a circuit. This saves routing area, as there are fewer connections to route between logic blocks. However, look-up table complexity grows exponentially with the number of inputs, so it is impractical to use a LUT with a large number of inputs as a logic block.

Instead of creating a larger logic block by increasing the number of inputs to a LUT, we can simply group several LUTs together and provide local routing to interconnect

Intel Exhibit 1006 Part 2
Intel v. Iida

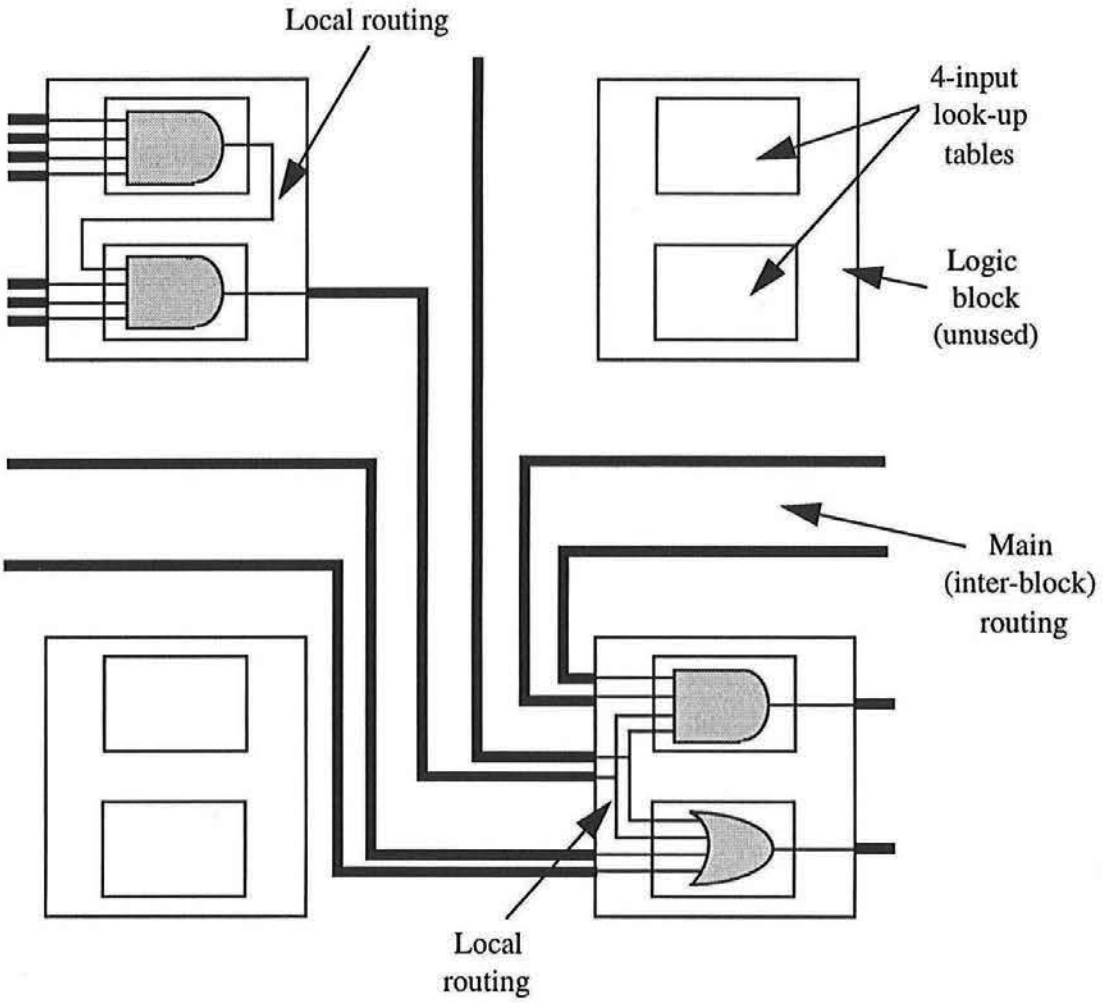


FIGURE 6.1 Implementation of a circuit in an FPGA with a cluster size of 2.

them. We call the resulting logic block a *logic cluster* [8, 9, 10]; the exact structure of a logic cluster was described in Section 3.1.1. Figure 6.1 shows a circuit implemented in an FPGA in which each logic cluster contains two four-input look-up tables. Notice that many connections are made via the local interconnect within a cluster.

One major advantage of an FPGA employing a logic block that contains several LUTs is that fewer logic blocks will be needed to implement a circuit than would be needed if each logic block was a single LUT. This reduces the size of the placement and routing problem considerably. Since placement and routing is usually the most time-consuming step in mapping a design to an FPGA, cluster-based logic blocks can significantly reduce design compile time. As FPGAs grow larger, it is important to keep this compile time from growing too large or one of the key advantages of FPGAs, rapid prototyping and design spins, will be lost [127].

Cluster-based logic blocks also have the potential to significantly improve FPGA speed. In FPGAs composed of logic clusters, many connections will be made via the local routing within a cluster. Since this local routing can be made faster than the general-purpose routing between logic blocks, cluster-based logic blocks can improve FPGA speed. It is not obvious which logic cluster size leads to the highest speed FPGA, however. On the one hand, larger logic clusters lead to a higher fraction of connections being made via local interconnect. On the other hand, as the size of a logic cluster increases, the local cluster interconnect becomes slower, potentially resulting in a speed decrease even though more connections are captured within the logic clusters.

The area impact of grouping multiple LUTs into a logic cluster is also complex. Grouping related LUTs together into a single logic block reduces the number of connections to be routed between logic blocks, which saves routing area. Since the general-purpose interconnect consumes most of the die area in SRAM-based FPGAs, this is a significant area savings. On the other hand, in the logic clusters we study the area required by the local routing within a cluster grows quadratically with cluster size. For sufficiently large clusters, then, the area used by this local interconnect will exceed the area saved in the general interconnect.

We explore four questions concerning the design of cluster-based logic blocks. First, how many distinct inputs should the FPGA routing provide to a cluster of LUTs? Reducing the number of inputs to a logic block saves routing area, but if the number of inputs is too low many circuits will be unable to use all the LUTs in a logic cluster, wasting area. Secondly, how should the flexibility of the logic block / routing interface (i.e. F_c [1]) change as the number of LUTs in a logic cluster changes? Third, how many LUTs should be included in a cluster to create FPGAs with the best combination of speed and area-efficiency? Finally, how is the time required to compile a circuit affected by the size of logic cluster used? Recent FPGAs from Xilinx, Altera, Lucent Technologies, Actel and Vantis have all grouped several LUTs together into a larger logic block, but there has been little published work investigating any of these questions.

Recall from Section 3.1.1 that we can describe a logic cluster with four parameters: the number of logic inputs (I), the number of BLEs (LUTs and registers) in a cluster (N), the number of clock inputs (M_{clk}), and the number of inputs to each LUT (K). In this chapter we fix the number of clocks per cluster at one for all our experiments, since the MCNC benchmark circuits we use to evaluate architectures all have only one clock. We set the number of inputs to each LUT, K, to 4, since previous research has shown LUTs of this size are the most area-efficient [25], and because this is the LUT size used in most commercial FPGAs. We will investigate logic clusters with

different values of I and N, however, as answering the questions posed above involves finding the most appropriate values of I and N.

6.2 Experimental Methodology

Our goal is to determine the cluster parameters that lead to the fastest and most area-efficient FPGA architectures. There are no detailed analytic models of FPGA architectures and circuitry, so we must evaluate architectures experimentally.

We implement a set of twenty benchmark circuits into each FPGA architecture of interest, and measure the circuit speed and area required for each architecture. We implement each circuit using an automatic CAD flow similar to that used by typical FPGA users: technology-mapping, placement and routing. The benchmark circuits used are the 20 largest MCNC circuits [136]; they range in size from 1064 to 8383 BLEs. These circuit sizes are typical of the designs being implemented in current commercial FPGAs.

6.2.1 CAD Flow

Figure 6.2 illustrates the CAD flow used in these experiments. First, the SIS [142] synthesis package is used to perform technology-independent logic optimization of each circuit. Next, each circuit is technology-mapped into 4-LUTs and flip flops by FlowMap and the Flowpack post-processing algorithm is used to optimize the mapping [46]. Our timing-driven T-VPack program (described in Section 3.1.3) then maps this netlist of 4-LUTs and flip flops into logic clusters with the specified values of N and I. At this point, then, the circuit is described as a set of interconnected logic blocks of the exact type that exist in the FPGA we're targeting. Finally, we use VPR to place and completely (combined global and detailed) route the circuit. In this chapter all routing was performed by the VPR timing-driven router described in Section 4.4.

As Figure 6.2 shows, the circuit is repeatedly routed with different channel capacities until VPR finds the minimum number of wire segments per channel required to successfully route the circuit, which we call W_{min}. FPGA manufacturers normally build enough routing into their FPGAs that "average" circuits have some spare routing available. We model this by performing a final "low-stress" routing of each circuit with the number of tracks per channel set to $1.2 \cdot W_{min}$. Our delay model then estimates the circuit critical path, and our area model estimates the total transistor area needed to lay out the FPGA. At the end of this CAD flow, then, we have enough

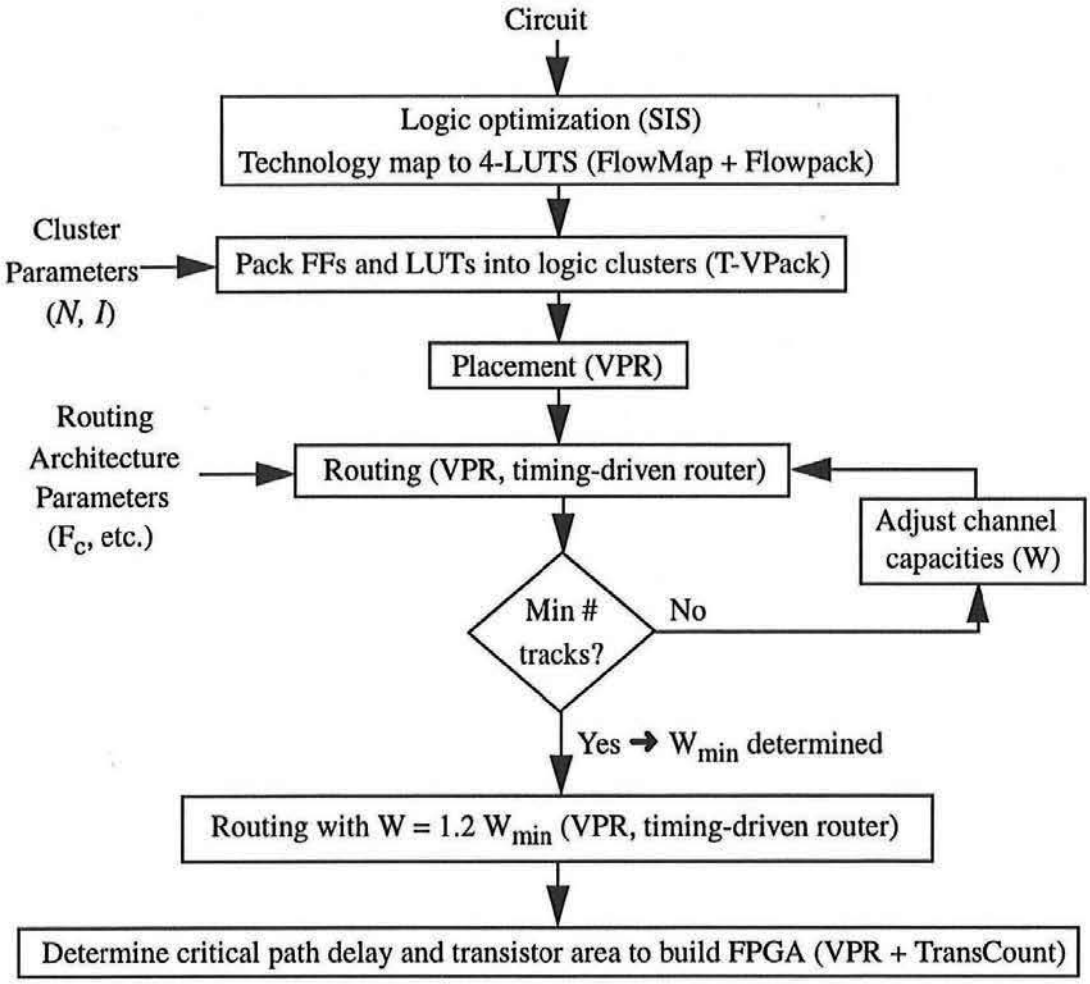


FIGURE 6.2 Architecture evaluation flow.

information to compare both the speed and the area-efficiency of one logic block architecture to another.

Notice that in the above CAD flow we are allowing the channel width to vary according to the needs of each circuit. By allowing the channel width to vary and searching for the minimum routable width, we can detect small improvements in FPGA architectures or CAD algorithms that might otherwise go unnoticed. If instead we mapped each circuit into an FPGA with a fixed channel width, we would only know whether each circuit successfully routed or not. It is more difficult to draw architectural conclusions from such a "binary" result.

6.2.2 Area Model

Our area model is based on counting the number of *minimum-width transistor areas* required to implement each FPGA architecture. A minimum-width transistor area is simply the layout area occupied by the smallest transistor that can be contacted in a process, plus the minimum spacing to another transistor above it and to its right, as shown in Figure 6.3. Since the area of typical commercial FPGAs is dominated by transistor area,[1] the most accurate way to assess the area of an FPGA architecture, short of actually laying out each FPGA architecture studied, is to estimate the total transistor area required by its layout. By counting the number of minimum-width transistor areas required to implement an FPGA, rather than the number of square microns which these transistors would occupy, we obtain a process-independent estimate of the FPGA area.

Some transistors in FPGAs require a drive strength greater than that of a minimum-width transistor. These transistors must either be made wider than minimum-width or

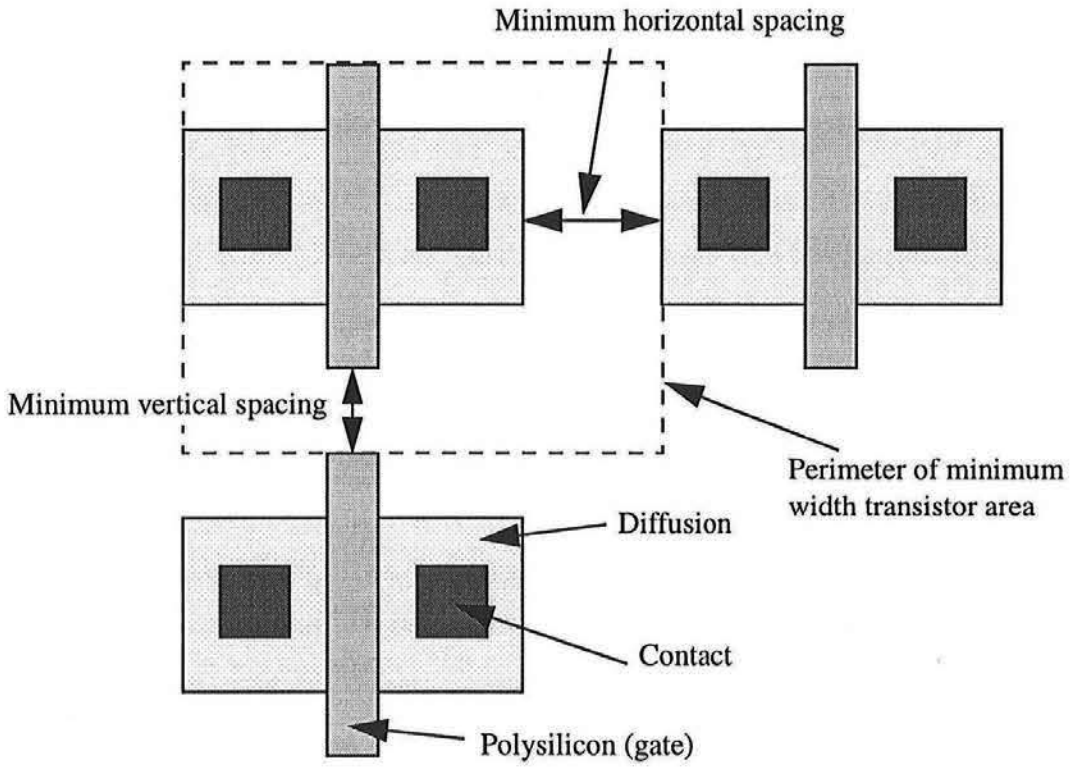


FIGURE 6.3 Definition of a minimum-width transistor area.

1. We have discussed this issue with FPGA architects at both Xilinx and Altera, and they have confirmed that transistor area determines the die size of their current FPGAs.

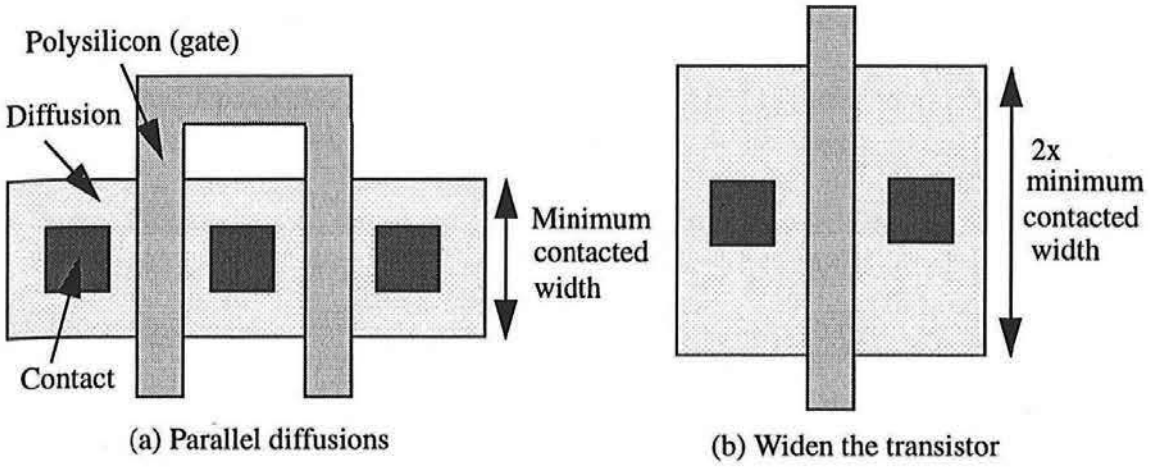


FIGURE 6.4 Methods to create a structure with 2x the drive strength of a minimum-width transistor.

their drive strength must be increased via parallel diffusion regions, as Figure 6.4 shows. The transistor active area increases in either case, but notice that if the parallel diffusions technique of Figure 6.4(a) is used, the transistor active area increases by less than a factor of two. As well, the spacing to the next transistor does not increase when a transistor's drive strength is increased, so the active area plus spacing area of a transistor with twice the minimum drive strength is less than two minimum-width transistor areas. We examined the layout rules from a TSMC 0.35 μm process and from an LSI Logic 0.4 μm process, and determined how much extra area was required to give transistors greater drive strength, either by making them wider or by paralleling diffusions. In both the LSI and TSMC processes, the number of minimum-width transistor areas required by a transistor, *trans*, (averaged over the different layout options) is:

$$\text{Minimum width transistor areas(trans)} = 0.5 + \frac{DriveStrength(trans)}{2 \cdot DriveStrength(MinimumWidth)}. \quad \textbf{(6.1)}$$

A transistor with the minimum drive strength therefore takes 1 minimum-width transistor area, while a transistor with double the minimum drive strength requires 1.5 minimum-width transistor areas.

In order to apply (6.1) to determine an FPGA's area, we must determine the number and size of the transistors required to build every structure in the FPGA. Appendix B.1 provides schematics showing how we build the key structures in an FPGA, and Section 6.2.5 discusses transistor sizing issues. In general, we tried to build an FPGA with as few transistors as possible without unduly compromising speed. We created a program, *TransCount*, that determines the area of a cluster-based logic block (includ-

ing the local cluster routing) with any values of N, I, K, and M_{clk}. This program is fairly sophisticated, and models such effects as buffer resizing as a function of the fanout of the connections within a logic block, and builds multi-stage buffers when high drive strengths are required. Of course the area of an FPGA includes not only the logic block area, but also routing area. VPR determines the routing area of each FPGA of interest, and by adding this area to the logic block area we obtain the total FPGA area. Recall that to evaluate the area of the FPGA needed to route a given circuit we build an FPGA with a channel width, W, of $1.2W_{min}$.

6.2.3 Delay Model

Our delay values are all based on the delays in TSMC's 0.35 μm, 3.3 V CMOS process. Some of the delays we use are listed in this section and in Appendix B.2, while some delays cannot be listed because the process information is proprietary and was obtained under a non-disclosure agreement.

To determine the critical path of a circuit, we must:

1. Determine the delay of every connection internal to a logic block,
2. Determine the delay of every connection between logic blocks, and
3. Perform a path-based timing analysis of the circuit using these delay values.

We found the delay of the connections within logic blocks by performing SPICE simulations of every structure in a logic block. Figure 6.5 shows the major structures and speed paths in a logic cluster, while Appendix B.1 contains transistor-level schematics that show how we build the multiplexers, buffers, look-up tables, and latches contained in a logic cluster. Since loading effects and input signal swing times can considerably change the delay of a circuit, we always simulated speed paths with their loads in place, and with the input to the path driven by the circuit which would drive it in a real FPGA. Notice that as the number of BLEs in a cluster increases, the number of inputs to the multiplexers forming the local cluster routing increases. As well, the fanouts (within the cluster) of the I cluster input signals and N cluster output signals increase, so the buffers driving these signals must be made larger (which results in a longer, and slower, inverter chain). For both these reasons, the delays of some of the paths within a logic cluster increase as the cluster size increases, as Table 6.1 shows.

After a routing is complete, we can perform step 2 above — determine the delay of every routed connection. The circuits we map contain thousands of nets, so SPICE simulation would be prohibitively time-consuming. Instead, we follow the procedure described in Section 2.2.4; we model pass transistors and buffers by equivalent circuits composed of resistors, capacitors, and idealized, constant delay elements. The values of the various equivalent resistances, capacitances and buffer intrinsic delays

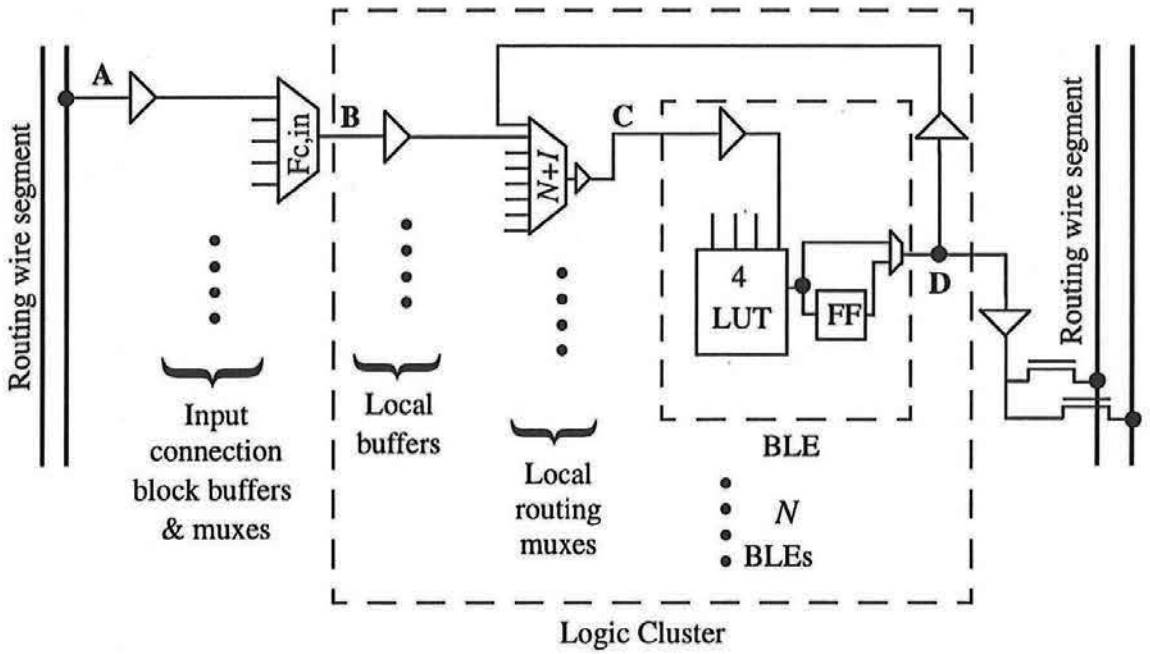


FIGURE 6.5 Structure and speed paths of a logic cluster.

TABLE 6.1 Important logic cluster delays in TSMC's 0.35 μm CMOS process.

Cluster Size (N)	A to B (ps)	B to C and D to C (ps)	C to D (ps)	B to D (ps)
1 (No local routing muxes)	760	55 (and no **D** to **C** path)	465	520
2	760	540	465	1005
4	760	675	465	1140
8	760	815	465	1280
16	760	970	465	1435
20	760	1000	465	1465

were again determined via SPICE simulations of the TSMC 0.35 μm process. For details of the procedure and circuit assumptions, see Appendix B. After a routing is complete, VPR uses these simplified models of pass transistors and buffers, as well as metal capacitance and resistance data, (all of which is specified in the architecture description file) to build an equivalent RC-tree for each net. It then computes the Elmore delay from the source to each of the sinks, as described in Section 4.5. We have found this linearized circuit model to be quite accurate: in Section 7.2.3 we

show that for a wide variety of different routing structures the delays computed by VPR are almost always within 9% of the delays computed by SPICE.

Finally, VPR performs a path-based timing-analysis using these delay values to determine the circuit critical path. Section 2.2.5 described the algorithms used in path-based timing analysis, and Section 4.5 described the VPR timing analyzer implementation.

6.2.4 Architecture Evaluation Metric: Area-Delay Product

One metric that we will use to evaluate the quality of different FPGA architectures is the area-delay product. This is a reasonable architecture metric for two reasons:

1. Intuitively, we want to find the point at which we are sacrificing the least amount of area for the most improvement in speed. Given that we can always trade area for speed (see below), and speed for area, it makes sense to combine these two factors into one curve to see where the best trade-off occurs.
2. The computational throughput of an FPGA (on a parallel algorithm) is simply the number of functional units multiplied by the clock speed. Another way of looking at this is, *throughput = (1/area per functional unit) · (1/delay)*. Therefore by minimizing the area-delay product, we maximize throughput.

There are two main factors which can affect the area-delay product of an FPGA: transistor sizing and the FPGA architecture. In general, the speed of an FPGA can be increased (to a point) by sizing up the buffers and transistors within the FPGA, but this increases area. Alternatively, the FPGA can be made smaller by sizing down the buffers and transistors, but this degrades the FPGA performance.

Throughout this chapter, we will size the transistors in each FPGA architecture to minimize the FPGA's area-delay product. Only by resizing transistors appropriately for each architecture in this way can we fairly compute the speed and area-efficiency of FPGAs with different logic block architectures.

6.2.5 FPGA Architectural Assumptions

To evaluate the speed and area of an FPGA we must choose not only the logic block architecture, but also a routing architecture and transistor sizes. The following sections detail all of our architectural choices.

Basic Architecture

We investigate island-style FPGAs in which each logic block is surrounded by routing channels on all four sides. The logic block input and output pins are evenly distributed around the logic block perimeter. Each circuit is mapped to the smallest square FPGA with enough logic blocks and pads to accommodate it.

Routing Architecture

We define the number of logic blocks which a routing segment spans as the logical length of that segment. In Chapter 7 we show that an architecture in which routing segments have a logical length of four, with 50% of the segments connected by tri-state buffers and 50% connected by pass-transistors, provides good area-efficiency and speed for FPGAs containing logic clusters of size four. This routing architecture is shown in Figure 6.6. We implicitly assume that this routing architecture is good for architectures containing logic clusters of all sizes, and we use this routing architecture in all of our experiments. Ideally, one would find the best routing architecture for each FPGA employing a different cluster size, but this would require a huge amount of effort. By basing all of our experiments on this routing architecture, we may slightly favor architectures with size four clusters over other architectures.

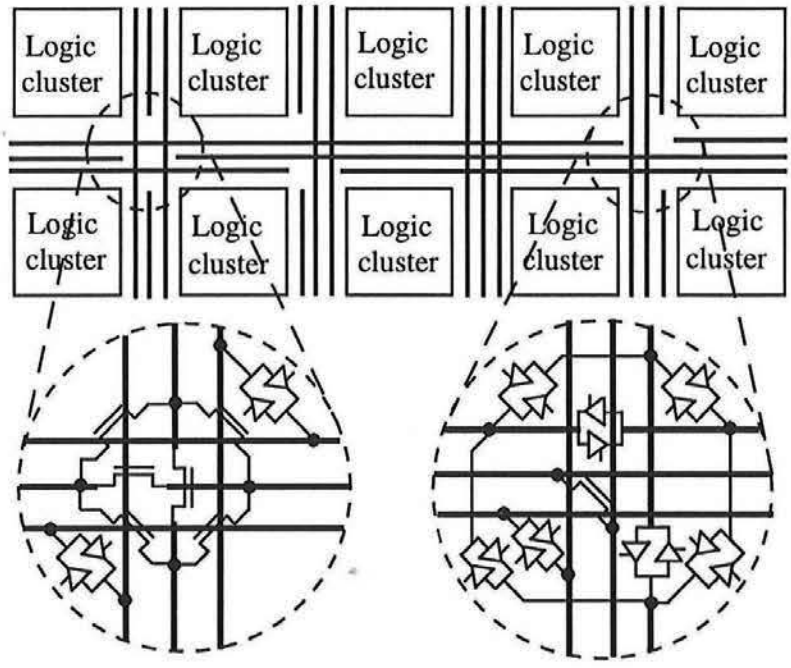


FIGURE 6.6 FPGA with length 4 segments, 50% buffered and 50% pass transistor switches.

Throughout this chapter we assume that metal all routing wires are laid out in metal 3, using the minimum width and spacing. See Appendix C.4 for a discussion of metal width and spacing issues in FPGAs.

Effect of Cluster Size on the Physical Length of FPGA Routing Segments

As we increase the cluster size, both the logic area per cluster and routing area per cluster grow. Figure 6.7 demonstrates how a tile (a logic block plus its associated routing) grows as cluster size is increased. This increased tile size results in routing segments with the same logical length having different physical lengths for logic clusters of different sizes.

We define the measured length of a routing segment as its physical length. The resistance and capacitance of a routing segment grow linearly with the segment's physical length. We have experimentally determined the average rate at which the FPGA tiles grow with cluster size, and have used this information to appropriately scale the routing segment resistance and capacitance values for the various cluster sizes. The increase in the resistance and capacitance of routing segments as the size, or granularity, of the FPGA logic block increases is an important effect that has often been neglected in prior FPGA architecture research.

Sizing Routing Transistors to Compensate for Different Physical Segment Lengths

To compensate for differences in the capacitance and resistance of routing segments in FPGAs using different sizes of logic clusters, we scale the routing pass transistors and buffers. All of our pass transistor and buffer scaling is in relation to a base archi-

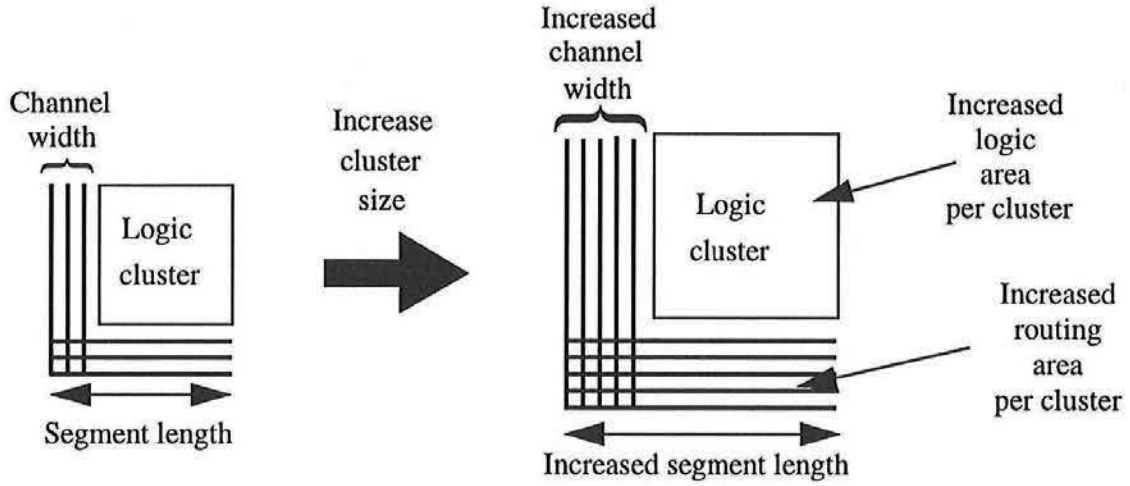


FIGURE 6.7 Effect of cluster size on physical length of routing segments.

tecture that has been area-delay optimized for clusters of size four. The method used to size routing transistors for this base architecture is described in Appendix C. From this base architecture, we linearly scale routing buffers and pass transistors depending on the relation between the new segment lengths and the base segment length. For example, in an FPGA with size 16 clusters, the physical segment length is approximately 2x longer than in an architecture with size 4 clusters. To maintain roughly the same speed per routing segment, we increase the size of the routing switches connecting to each wire by a factor of 2. In Section 6.5 we verify that this linear scaling of buffers and pass-transistors with physical segment length provides good results.

In our architecture models, we account for variations in delay caused by resizing buffers and pass-transistors. Also, changes in area due to the use of different sizes of routing pass-transistors and buffers are automatically calculated by VPR.

6.3 Cluster Inputs Required vs. Cluster Size

As discussed in Section 6.1, the first question we wish to answer is how many distinct inputs, I, should be provided to a cluster of size N. Since the number of transistors required to implement each of the multiplexers in the cluster local routing (see Figure 3.1(b)) grows linearly with I (for large I), we would like to make I as small as possible. On the other hand, if I is made too small, many of the BLEs in a logic cluster may become essentially unusable, reducing logic utilization and wasting area. We find the minimum value of I that allows good cluster utilization by running benchmark circuits through the first two steps shown in Figure 6.2, technology-mapping and cluster packing, and measuring the resulting logic utilization for different values of I. We define logic utilization to be the average number of BLEs per cluster that a circuit is able to use divided by the total number of BLEs per cluster, N.

Figure 6.8 shows how the average logic utilization of our 20 benchmarks varies with I for three different logic cluster sizes. The horizontal axis is the number of distinct inputs to the cluster relative to the total number of BLE inputs in a cluster, i.e. $I/(4N)$. For very low values of I, the logic utilization is very low, as one would expect. It is interesting, however, that when I is only 50 to 60% of the total number of BLE inputs, the logic utilization is essentially 100%. Clearly it is possible to pack BLEs together so that they have many common inputs and can reuse locally generated outputs. The relative amount of input sharing and output reuse increases slightly with logic cluster size, causing the curves in Figure 6.8 to shift to the left as cluster size increases.

The solid line in Figure 6.9 shows the value of I required to achieve 98% logic utilization as the cluster size, N, is varied, while the dashed line shows how the average

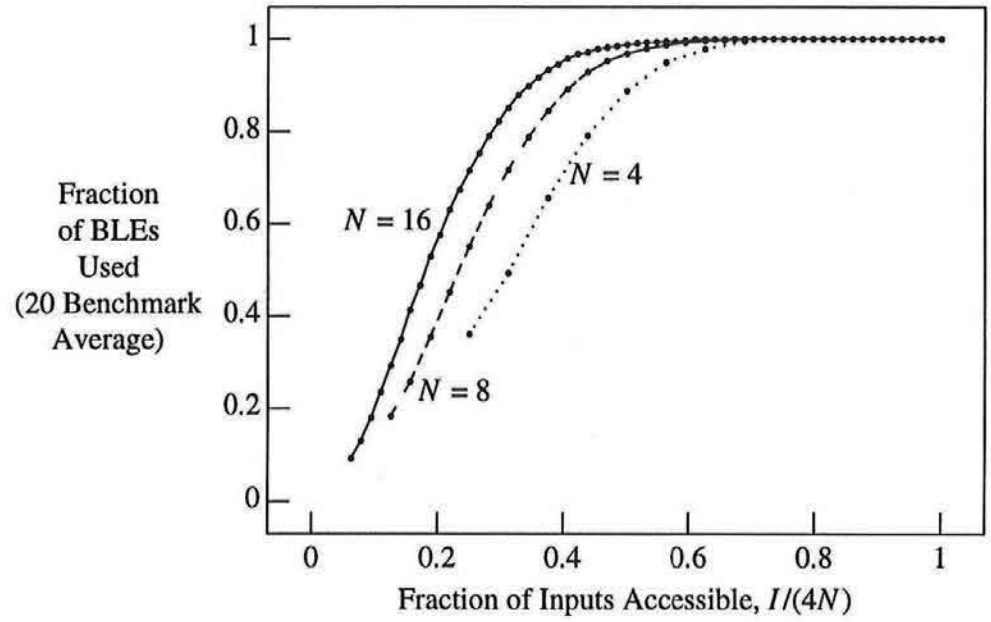


FIGURE 6.8 Logic utilization vs. number of logic cluster inputs.

number of logic cluster inputs that are actually used varies with cluster size. Although there are $4N$ BLE inputs in a logic cluster of size N, the number of inputs required to achieve 98% logic utilization is approximately $2N + 2$. Furthermore, the average number of logic cluster inputs that are actually used grows even more slowly. On average, a cluster of size 1 uses 3.57 of its inputs, while an cluster of size 20 uses only 25.2 of its inputs. In other words, while the logic per cluster has increased by a

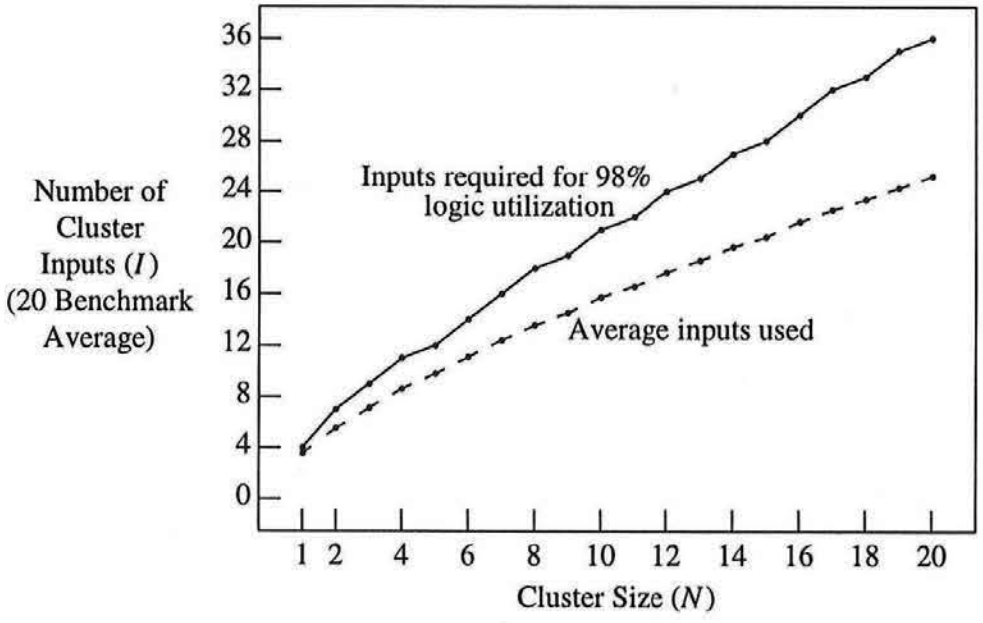


FIGURE 6.9 Variation in inputs required and inputs used with cluster size.

factor of 20, the average number of connections that must be routed to each cluster has increased by a factor of only 7.

Our results indicate that commercial FPGAs can be more aggressive in reducing the value of I. For example, the Altera Flex 8K FPGAs use logic clusters with $N = 8$ and $I = 24$, while our results indicate that $I = 18$ suffices for a cluster of this size. Similarly, the Xilinx 5200 FPGA uses a logic block very similar to a logic cluster with $N = 4$, and makes all 16 LUT inputs accessible, while our results suggest 10 to 11 inputs are sufficient. Reducing I in this manner simplifies the local routing multiplexers within a logic cluster and reduces the number of logic block pins that must be connected to the FPGA routing, resulting in considerable area savings.

6.4 *Flexibility of Logic Block to Routing Interconnect vs. Cluster Size*

Before we can apply the experimental flow of Section 6.2 to see how area-efficiency varies with cluster size, we must choose F_c, the number of routing tracks to which each logic block pin can connect. On the one hand, using a smaller value of F_c reduces the number of programmable switches in the FPGA routing, which improves area-efficiency. On the other hand, smaller values of F_c make an FPGA less routable so that larger channel capacities, W, will be required to successfully route circuits. This reduces area-efficiency by increasing the routing area. The goal is to choose a value of F_c that balances these two competing objectives and achieves good area-efficiency.

For a cluster of size 1, our experiments and those of Rose and Brown [30] have shown that a good value of F_c is W; i.e. each logic block pin can be connected to any routing track in an adjacent channel. For larger clusters, however, setting F_c to W provides far more routing flexibility than is required, wasting area.

Recall from Section 3.1.1 that the logic clusters we investigate are *fully-connected*. In other words, a BLE input can be connected to any cluster input or to the output of any of the BLEs within the cluster via the local routing. In a fully-connected logic cluster all the cluster inputs and all the cluster outputs are *logically-equivalent*. That is, all of the inputs are functionally identical, and all of the outputs are functionally identical. This means that a net which is an input to a cluster can be connected to *any* of the I cluster inputs, and a net which is driven by a cluster output can be connected to *any* of the N cluster outputs. Changing the connections made by the multiplexer-based local routing can compensate for any pin swapping performed on the cluster input pins by the router. Changing which BLE within the cluster generates each of the functions

required by the netlist can compensate for any pin swapping performed by the router on the cluster output pins. Therefore the router has a great deal of flexibility in how it routes inter-cluster nets.

The logical equivalence of cluster inputs and of cluster outputs means that keeping F_c fixed at W, regardless of the cluster size, *N*, results in an excessive number of ways to connect to large logic clusters. For example, a cluster of size one has 4 inputs and one output. If $F_c = W$, then, there are 4W ways to connect to a cluster input and W ways to connect to the cluster output. A cluster of size 20, on the other hand, has 36 inputs and 20 outputs, so there are 36W ways to connect to a cluster input and 20W ways to connect to a cluster output if $F_c = W$. This excessive routing flexibility for a cluster of size 20 wastes a large amount of routing area, since we have added many more programmable switches to the routing than is necessary.

We have experimentally found that a more appropriate level of routing flexibility results when the F_c value for logic block output pins, $F_{c,output}$ is set to W/N, and all the experiments in the next section use this value. This choice of $F_{c,output}$ means that each of the W routing tracks can be driven by one output pin on each logic block, ensuring that all the routing tracks in a channel can be readily used to interconnect blocks. Since connections to logic block input pins require less area than connections to logic block output pins (as described in Appendix B.1), it is best to decrease $F_{c,input}$ more slowly than $F_{c,output}$ as the cluster size increases. We reduce $F_{c,input}$ from W to 0.2W as the cluster size increases from 1 BLE to 20 BLEs.

6.5 Speed and Area-Efficiency vs. Cluster Size

We are now in a position to examine which cluster size leads to FPGAs with the best speed and area-efficiency. Throughout this section, the number of inputs, *I*, to a cluster of size *N* is chosen to be the minimum value that allows T-VPack to achieve 98% logic utilization. This value of *I* allows our logic clusters to be essentially fully utilized, while minimizing the complexity of the cluster input multiplexers and the number of logic block pins to be connected to the main FPGA routing. We ran 20 benchmark circuits through the experimental flow described in Section 6.2, and determined the area-efficiency and speed of each in FPGA architectures employing different logic cluster sizes. We present only the average results across all 20 circuits in the benchmark suite; the results for individual circuits track the average quite closely.

In Figures 6.10 and 6.11, we show the geometric average over the benchmark circuits of the total FPGA area required and the critical path delay, respectively. Note that we are showing three different routing transistor sizings in each of these graphs to ensure

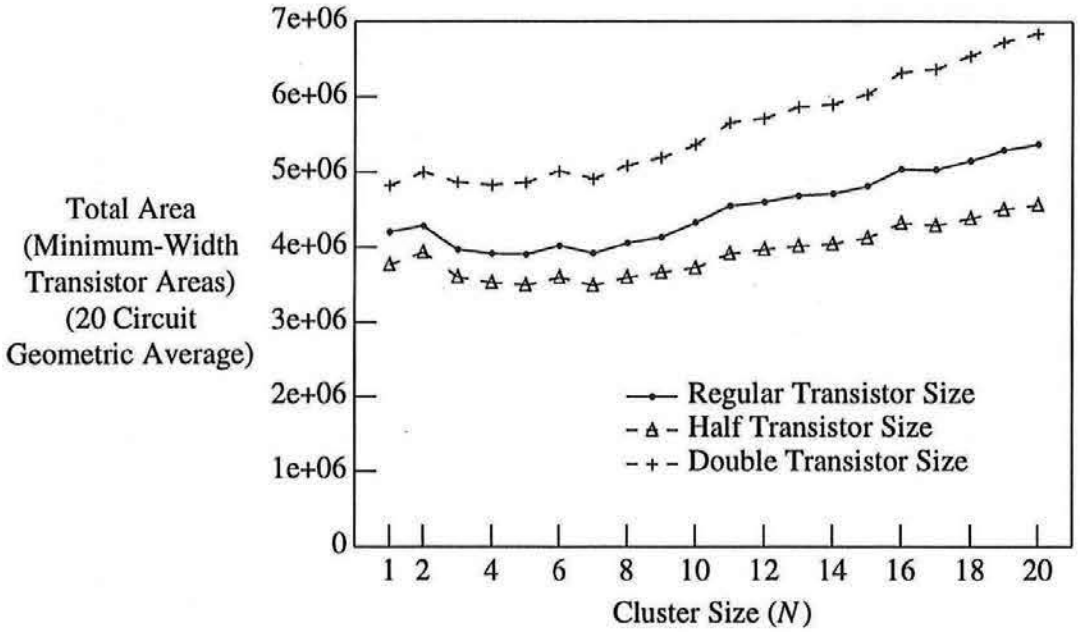


FIGURE 6.10 Total area vs. cluster size.

that we do not unfairly penalize any architecture with an inappropriate transistor sizing. The solid curves show the area and delay when we use the "normal" transistor sizing described in Section 6.2.5, while the dashed curves show the results when we use transistors that are one-half or double the size of those in the "normal" case.

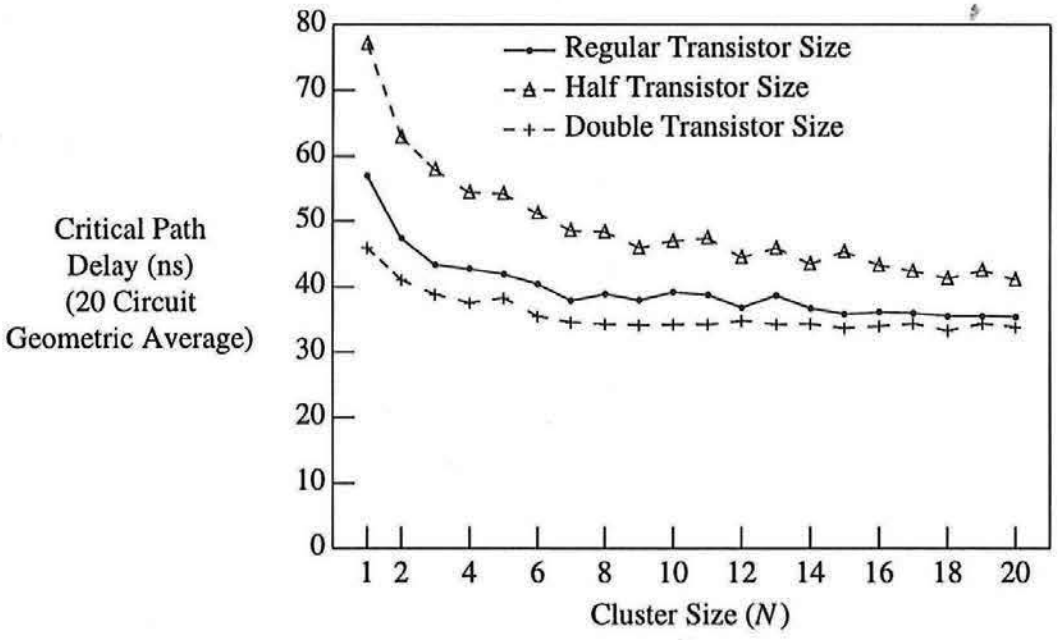


FIGURE 6.11 Critical path delay vs. cluster size.

Notice that we can indeed trade speed for area by resizing routing transistors — the half transistor size results have the smallest area, but the highest delay, while the double transistor size results have the smallest delay, but the highest area.

In Figure 6.10, all three curves show the same basic trend. Area initially increases as one moves from a cluster of size 1 to a cluster of size 2. This area increase occurs because a logic cluster of size 1 contains no local routing (it is a single BLE), while a cluster of size 2 does. The addition of this local routing to the FPGA requires a considerable amount of area, and at a cluster of size 2 it has not yet reduced the number of connections to route between clusters enough to compensate. Further increases in cluster size, to $N = 3$ and 4, improve area-efficiency because the local routing is able to more significantly reduce the amount of routing required between logic blocks. As the cluster size rises past $N = 8$ to 10, area-efficiency gradually degrades. The complexity of the local routing in a logic cluster grows quadratically with cluster size, and for sufficiently large clusters this swamps the area improvements gained by reducing the routing required between logic blocks.

Figure 6.11 shows that circuit speed increases significantly as we increase the cluster size. As one increases the cluster size from size 1 to 7, the circuit speed rapidly increases — with the "normal" transistor sizing, a size 7 logic cluster leads to circuits which are 51% faster than those implemented with a size 1 cluster. Increases in cluster size past $N = 7$ produce smaller incremental speed gains. For example, with the "normal" transistor sizing, a cluster of size 20 is 7% faster than a cluster of size 7.

In Figure 6.12, we show how the geometric average of the area-delay product achieved by the benchmark circuits varies with cluster size, again for three different transistor sizings. Notice that the "normal" transistor sizing provides the best area-delay product for all the architectures except a cluster size of 1, indicating that linearly scaling routing transistor size with the length of a layout tile is a good method to size transistors. For a cluster of size 1, however, the normal transistor sizing is smaller than optimal, and the double transistor size FPGA has a 7.5% lower area-delay product than the normal transistor size FPGA. There is a broad minimum in the area-delay product for cluster sizes from 4 to 10. A cluster of size 7 has the lowest area-delay product, but any cluster size between 4 and 10 is within 12% of the minimum, and hence would be a reasonable choice. Notice that moderate-size logic clusters significantly improve the area-delay product of an FPGA vs. using a single BLE logic block. Comparing a size 7 logic cluster (with the normal transistor sizing) to a size 1 logic cluster (with double-sized transistors — the best for this cluster size), one sees that the size 7 logic cluster has an area-delay product that is 33% lower than that of a size 1 cluster. An FPGA using a size 7 logic cluster is simultaneously 21% faster (a 17% delay reduction), and requires 19% less area than an FPGA using a size 1 logic cluster.

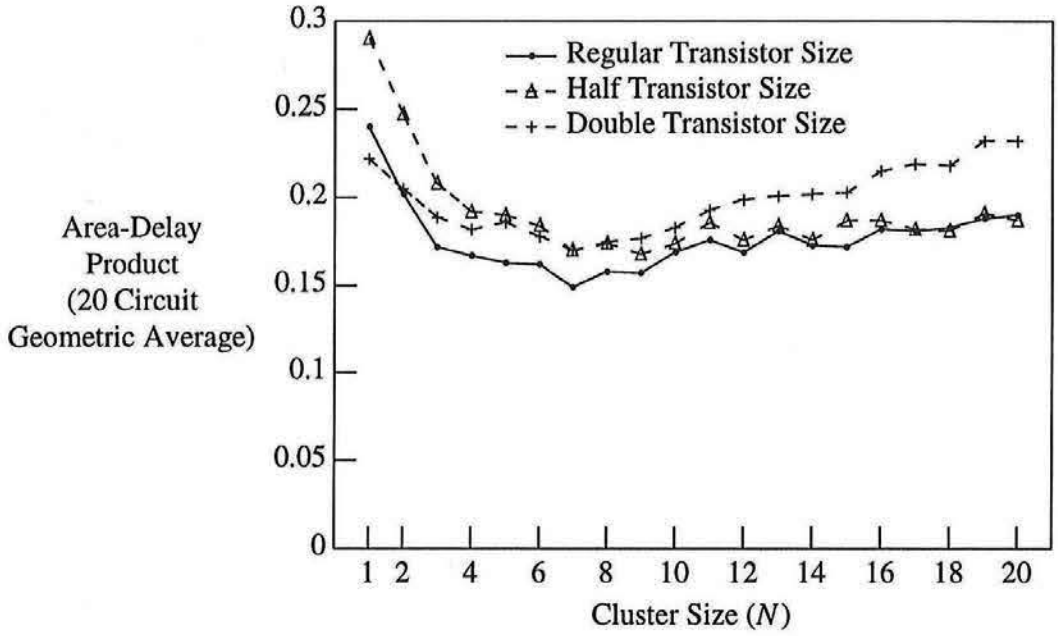


FIGURE 6.12 Area-delay product vs. cluster size.

6.5.1 Discussion of Delay vs. Cluster Size Results

In Figure 6.13 we show the relationship between the number of intra-cluster (fast) and inter-cluster (slower) connections on the critical path as a function of cluster size. As cluster size is increased, the number of intra-cluster connections on the critical path increases, and the number of inter-cluster connections decreases. This provides a circuit speedup since intra-cluster connections are faster than inter-cluster connections.[1]

Interestingly, the number of inter-cluster nets on the critical path does not decrease as much with cluster size as the inter-cluster delay decreases with cluster size (see Figure 6.14). From size 2 to size 20 we have a reduction in the number of inter-cluster nets on the critical path of 13% (Figure 6.13); compare this to the inter-cluster component of the critical path delay, which has been reduced by 39% over the same range (Figure 6.14). This means the circuit speedup visible in Figure 6.14 for larger cluster sizes is not only caused by a reduction in the number of inter-cluster connections on the critical path, but also by *inter-cluster connections on the critical path becoming faster.*

1. As cluster size is increased, intra-cluster multiplexer, buffer and wiring delays increase. If we were to increase the size of cluster to very large values, this effect would eventually result in intra-cluster delays becoming large enough that any gains obtained by making connections local to the cluster would be lost.

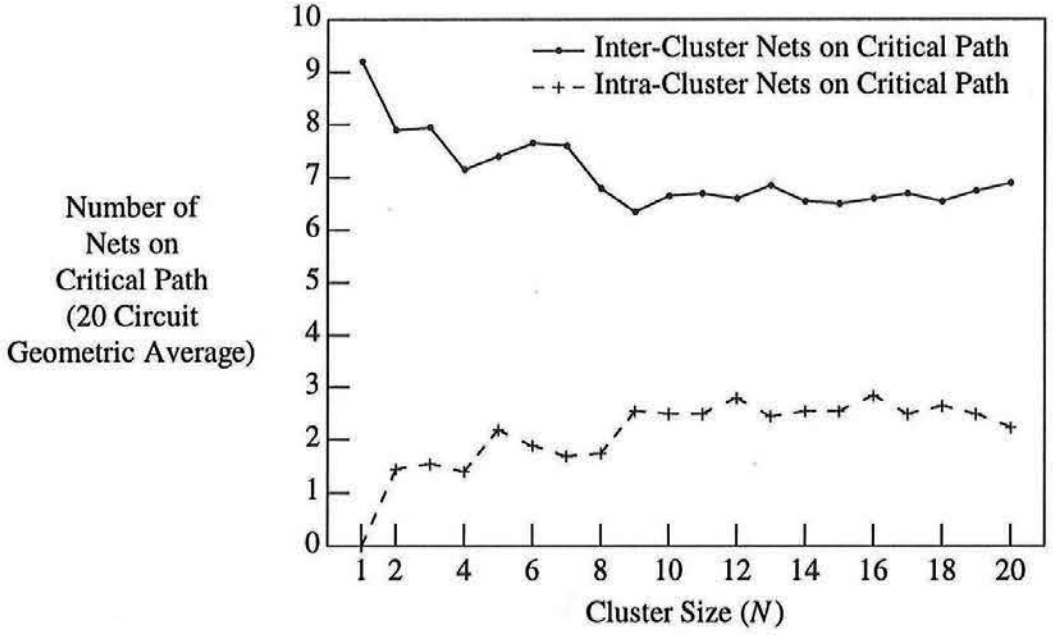


FIGURE 6.13 Inter- and intra-cluster nets on the critical path.

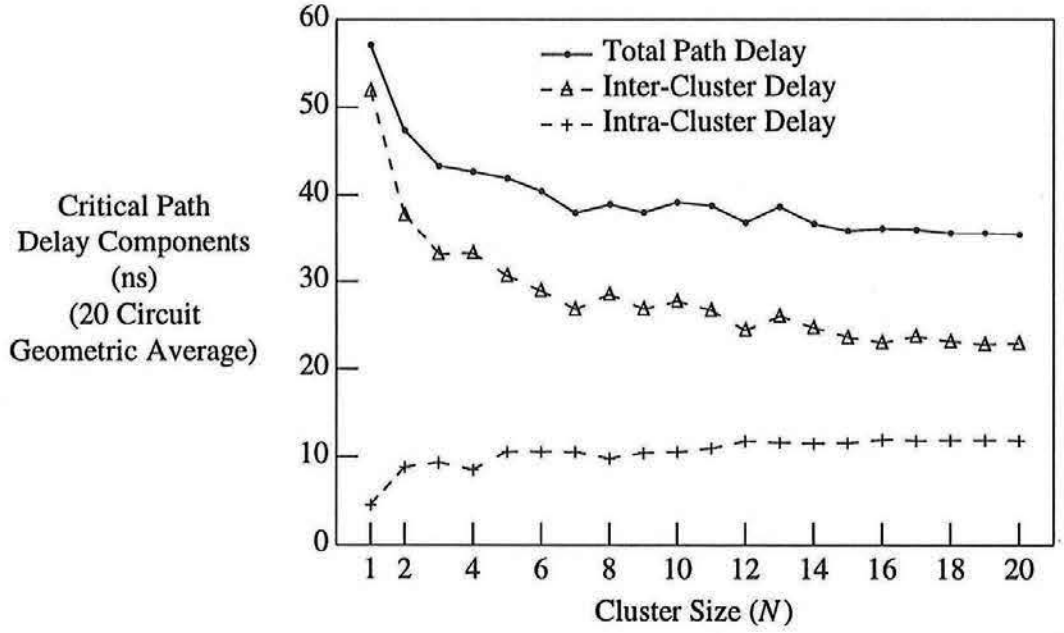


FIGURE 6.14 Breakdown of critical path delay into inter- and intra-cluster components.

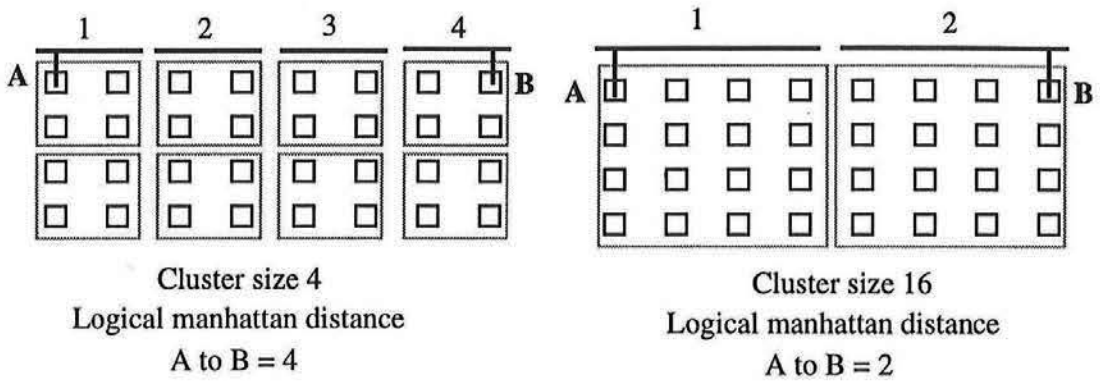


FIGURE 6.15 Decrease in logical manhattan distance as cluster size increases.

The improvement in inter-cluster delay with increased cluster size is caused primarily by a reduction in the "logical" manhattan distance spanned by connections in the FPGA, as shown in Figure 6.15. By sizing the routing pass transistors and buffers[1] to compensate for the increased physical length of routing wire segments associated with larger clusters, the delay of each routing segment has remained roughly constant. Since the total number of routing segments on the critical path has decreased due to the reduction of the "logical" manhattan distance, the result is a greater improvement in circuit delay than the reduction in the number of inter-cluster nets on the critical path would indicate.

6.6 Effect of Cluster Size on Compile Time

As the logic capacity of FPGAs increases and FPGAs capture larger and larger designs, compile times are increasing and threatening a key FPGA advantage: fast design spins. Consequently, FPGA architectures that lead to reduced circuit compile times are clearly desirable.

Figure 6.16 shows how the average CPU time (on a 300 MHz UltraSparc workstation) required to implement circuits varies with cluster size. The solid line in Figure 6.16 shows the total (packing, placement plus routing) compile time, while the two dashed lines show the placement and routing components of this compile time. The logic block packing time required is not significant — on average, T-VPack requires only about 1 second to pack a benchmark circuit.

1. Changes in delay and area due to different size routing buffers and pass transistors are accounted for in VPR's timing and area models.

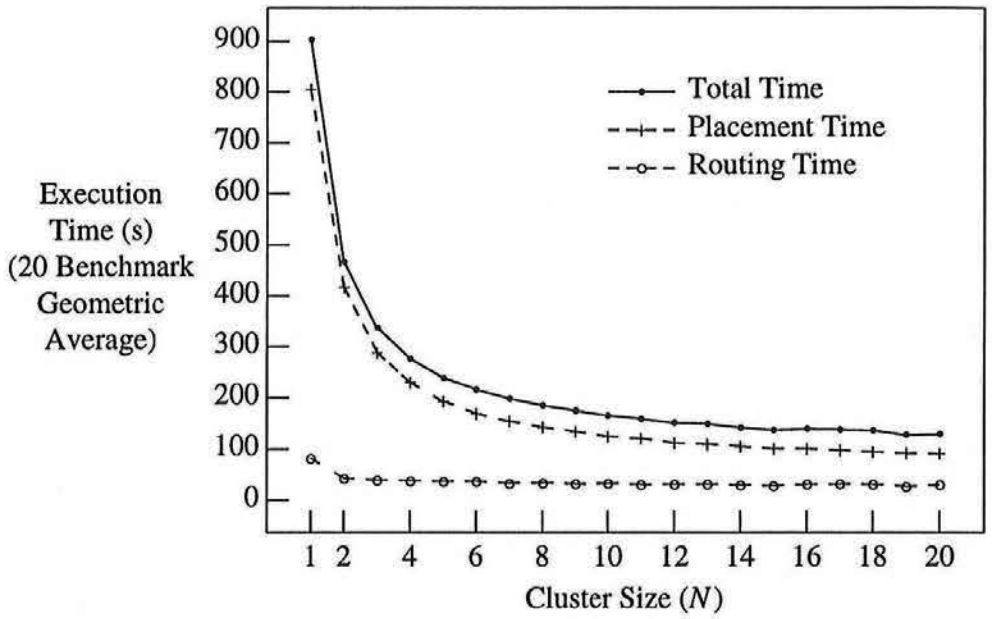


FIGURE 6.16 Variation of circuit compile time with logic cluster size.

Clearly, as larger logic clusters are employed in an FPGA the time to compile circuits is dramatically reduced. Building an FPGA with a size 20 logic cluster reduces the total CPU time required for placement and routing by *7x* vs. a size 1 logic cluster. As the number of BLEs per logic block increases, a circuit can be implemented with fewer logic blocks. Since the size of a placement problem is proportional to the number of logic blocks required to implement a circuit, this dramatically reduces placement time. In Figure 6.16, for example, one can see that the placement time is reduced by a factor of 8.8x as the cluster size increases from 1 to 20. Larger logic clusters also reduce the routing time. Since more connections are made via the local cluster routing in larger clusters, there are fewer inter-cluster connections to route. For example, using a size 20 logic cluster reduces routing time by 2.7x vs. using a size 1 cluster.

6.7 Summary

There are four main conclusions to be drawn from this work. First, the number of distinct inputs required by a logic cluster grows fairly slowly with cluster size, N. A cluster of size N requires approximately $2N + 2$ distinct inputs (for $N \leq 20$). Second, because all the input and output pins of a cluster are logically equivalent, one can significantly reduce the number of routing tracks to which each logic cluster pin can connect, F_c, as one increases the cluster size. We have found that a good value for

$F_{c,output}$ is W/*N*; it is best if the $F_{c,input}$ value is somewhat higher. Third, logic clusters containing between 4 and 10 BLEs all achieve good performance (as measured by their area-delay product) so any cluster in this range is a reasonable choice. Logic clusters in this size range achieve significant area and speed improvements relative to a single BLE logic block. For example, an FPGA employing a size 7 logic cluster requires 19% less area, achieves 21% higher speed, and has an area-delay product 33% lower than an FPGA using a single BLE logic block. Finally, large logic clusters significantly reduce design compile time. Size 7 and 20 logic clusters reduce placement and routing time by 4.6x and 7x compared to a single BLE logic block, respectively.

CHAPTER 7

Detailed Routing Architecture

In this chapter we explore a series of *detailed routing architectures* to find which ones lead to the best FPGA area and speed [14]. The detailed routing architecture of an FPGA specifies the length of every wire in the FPGA, the type of switch used to make every connection, the switch block topology, the metal width and spacing of each routing wire, and several other related parameters. In the next section we more precisely define all the parameters determining an FPGA's detailed routing architecture, and explain why detailed routing architecture issues are so crucial in FPGA design. Section 7.2 then describes the experimental flow we use to evaluate different routing architectures.

In Sections 7.3 through 7.7 we employ this flow to explore several key detailed routing architecture issues. We begin in Section 7.3 by investigating the best switch block topology and best F_c values for FPGA architectures that include wires that span more than one logic block. We also determine the best routing wire length when all wires in the FPGA have the same length in Section 7.3. In Section 7.4 we investigate FPGAs that contain two different lengths of routing wires, and a mix of pass-transistor and tri-state buffer routing switches. In Section 7.5 we evaluate the utility of internally depopulating wire segments, and determine the best amount of depopulation. Section 7.6 examines the speed gains attainable by increasing the spacing between some or all of the routing wires. Finally, Section 7.7 provides an overview of all the architectures examined in this chapter, comparing the speed and area achieved by the best architectures against a routing architecture similar to that of the Xilinx XC4000X series FPGAs.

7.1 Motivation

Most circuit delay in FPGAs is due to routing delays, rather than logic block delays, and most of an FPGA's area is devoted to routing [1]. In many ways, then, detailed routing architecture is *the key* architectural issue in FPGAs. When we refer to the *detailed routing architecture* of an FPGA we are referring to the values of a very wide range of parameters, including:

- The number of wires, F_c, to which each logic block input pin or output pin can connect,
- The switch block topology (which defines which wires can connect at a switch block),
- The segmentation distribution; that is, the different lengths of wire segments in the FPGA, and the fraction of tracks in a channel that are composed of wires of each length,
- The switch-block internal population and connection-block internal population of each wire segment,
- The type of switch (pass transistor or tri-state buffer) used to connect each routing wire to other wires,
- the sizes of the transistors used to build the various programmable switches, and
- the metal width and spacing of the various routing wires.

Recall that definitions of all the parameters listed above were provided in Section 2.1.3, and Section 4.2.1 provided a new and more general definition of internal switch population. The first four parameters above are topological parameters that define the connectivity of the FPGA routing architecture; the last two parameters are purely electrical, and the type of switch used to make each connection is both a topological and an electrical parameter.

Every parameter listed above is important, and the choice of each involves balancing complex trade-offs and interactions with other parameters. For example, if one chooses a segmentation distribution with too many short wires, long connections will have to be constructed using several short wires connected in series, resulting in poor speed. If the segmentation distribution includes too many long wires, however, some short connections will be forced to use long wires, degrading speed and wasting area. Similarly, an architecture with too many or too few tri-state buffer routing switches will clearly be suboptimal. Pass transistors require less area, and they are faster than buffers for short connections, but connections that pass through many switches are better served by tri-state buffers. As well, the best mix of routing switches is dependent on the segmentation distribution. A segmentation distribution with many long

wires will rarely have to connect many switches in series to make a connection. Consequently, an FPGA with many long wires can use a higher fraction of pass transistor-based switches in its routing than an architecture with a shorter segmentation distribution.

Previous research into detailed routing architecture (as described in Section 2.1.3) has generally focused on only one or two of the parameters listed above, and neglected the others. In this work we have tried to take a more holistic approach, investigating appropriate values for all of these parameters, and determining important parameter interactions. In addition, some important parameters have never been investigated before. For example, all prior studies have assumed that every routing switch is a pass transistor — the possibility of some routing switches being tri-state buffers has not been considered. Similarly, no prior work has considered varying the metal width and spacing of the routing wires. The internal population issue has been investigated only in a very limited way [21] — the only question addressed to date has been whether or not to completely depopulate length 3 wires. Similarly, while considerable work has been conducted on finding the best switch block topologies for FPGAs that contain only length 1 wires, there has been no research into the best switch blocks for use with FPGAs that contain longer wires.

In addition to examining a broader spectrum of FPGA architectures than prior researchers, we have also taken considerable care to create accurate area and delay models. Simplified and abstracted models can lead to questionable conclusions, particularly when we can compare such a vast array of widely different FPGAs. Consequently, we have used the "true" delay metric — critical path delay — and a highly detailed area model based on estimating the total transistor area required by a routing architecture.

Finally, in this study we spent considerable effort to ensure our CAD tools properly optimized for every routing architecture in the entire spectrum investigated. Hence we created a router that was both routability- and truly timing-driven (as described in Section 4.4) and understood issues such as segment lengths, buffer insertion, and metal capacitance and resistance. It is the flexibility of the VPR timing-driven router, as well as the flexibility of our area and delay models, that have enabled us to investigate such a broad spectrum of architectures.

7.2 *Experimental Methodology*

We explore different architectures by implementing the twenty largest MCNC benchmark circuits (which range in size from 1064 to 8383 BLEs) into each FPGA archi-

tecture of interest. We implement each circuit with an automatic CAD flow similar to that used by FPGA users: technology-independent logic optimization, technology-mapping, placement and routing. We then compare the circuit delay achieved and the area required by each architecture.

In the next section we describe the portions of the FPGA architecture, such as the logic block used, that are held constant throughout all the experiments of this chapter. The subsequent sections detail the CAD flow used to implement circuits in each FPGA, the area and delay models used to evaluate the result quality achieved by the various architectures, and the "experimental philosophy" that governed how we explored the huge detailed routing architecture design space.

7.2.1 FPGA Architectural Assumptions

In this chapter we are investigating different detailed routing architectures, so we hold the other architectural parameters, such as the logic block used, constant throughout the experiments. All the FPGAs we investigate are island-style FPGAs (described in Section 2.1.3), and each channel contains the same number of tracks and has the same segmentation distribution.

The logic block of all the FPGAs studied in this chapter is a logic cluster of four BLEs, with ten inputs and one clock. Using the notation of Section 3.1.1, then, the logic block is a cluster with $N = 4$, $I = 10$, $K = 4$ and $M_{clk} = 1$. This logic block is similar in size to the logic blocks employed in recent commercial island-style FPGAs [145, 13]. As well, in Section 6.5 we showed that this logic block has good speed and area-efficiency. The input and output pins are evenly distributed around the perimeter of the logic block, since the results of Section 5.3.1 showed this pin positioning is best.

The number of I/O pads that fit into the height or width of a logic block is set to four, in line with the relative sizes of pads and 4-LUTs of current FPGAs [4, 5, 7]. With this assumption three of the twenty benchmark circuits we use (bigkey, des and dsip) are pad-limited. We always map each circuit to the smallest square logic block array that has enough logic blocks and pads to accommodate it. Since commercial FPGAs normally distribute the circuit clock through a special, dedicated routing resource, we do not route the clock net in sequential circuits.

Finally, just as in Section 6.2.5, the size of the transistors used in the routing switches is a key architectural issue. We believe it is most appropriate to size these transistors to achieve the minimum area-delay product of the resulting routing resources. Appendix C details the transistor sizing experiments we conducted with the TSMC 0.35 μm process to find the best routing transistor sizes for use in an FPGA employ-

ing a size 4 logic cluster as its logic block. In Appendix C, we show that for a very wide range of routing wire segment lengths, tri-state buffers achieve the best area-delay product when their nMOS transistors have five times the drive strength of a minimum width nMOS transistor. To achieve equal rise and fall times, the pMOS transistors used in these buffers must be larger — 9.5 times the drive strength of a minimum width pMOS transistor. For routing switches composed of pass transistors, the best area-delay product is achieved when the transistors have ten times the drive strength of a minimum-width nMOS transistor. We use these routing transistor sizings throughout this chapter. Appendix C.4 also shows that to minimize the area-delay product of an FPGA one should use minimum width routing wires. Consequently all the results in this chapter assume minimum-width metal for routing wires.

7.2.2 CAD Flow

The CAD flow we use to evaluate routing architectures is almost identical to that of the previous chapter, as Figure 7.1 illustrates. SIS [142] and FlowMap [46] are used to optimize each circuit's logic and technology map each circuit into 4-LUTs and reg-

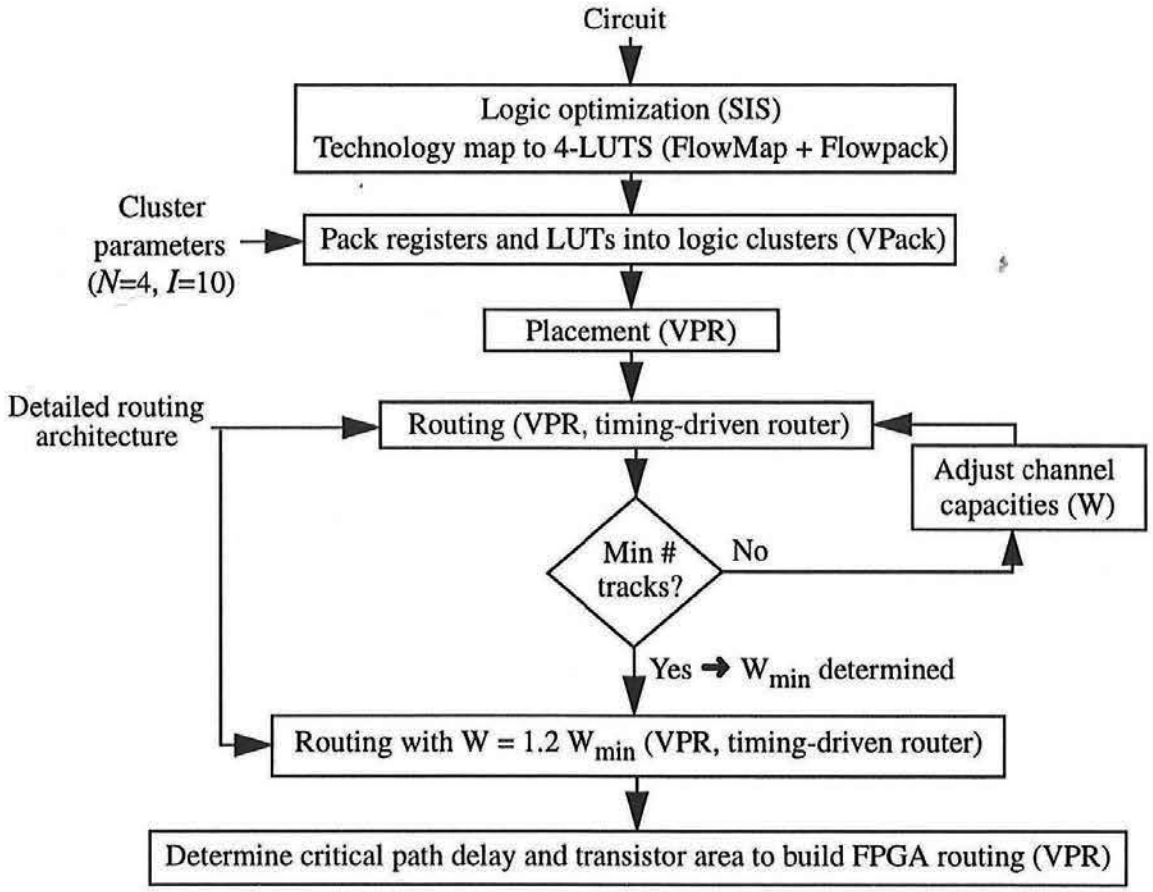


FIGURE 7.1 Architecture evaluation flow.

isters, respectively. VPack (described in Section 3.1.2) then groups these 4-LUTs and registers into logic clusters of the desired size (4 BLEs per cluster, using no more than 10 distinct inputs). While the timing-driven T-VPack tool is a superior to VPack, it was not completed at the time this study was done. Since throughout this chapter we always use the same logic block and vary only the routing architecture, however, it is not crucial that the logic block packing be the best possible. In other words, using T-VPack instead of VPack would make the circuits somewhat (~10%) faster in *every* routing architecture we evaluate, but should not affect the relative "goodness" rankings of the various routing architectures.

Once logic block packing is complete, VPR then places the circuit, and the VPR timing-driven router is repeatedly invoked with different channel capacities to determine the minimum number of tracks per channel, W_{min}, required to route the circuit. Since FPGAs are normally built with enough routing that "average" circuits have some spare routing available, we perform a final "low-stress" routing of each circuit with the number of tracks per channel set to $1.2 \cdot W_{min}$. We then apply our delay model to estimate the delay of the circuit critical path, and our area model to estimate the total transistor area needed to lay out all the routing in this FPGA. At this point, then, we have enough information to compare both the speed and the area-efficiency of one routing architecture to another.

7.2.3 Delay Model Accuracy

Our delay model was described in detail in Section 6.2.3. Recall that we model pass transistors and buffers in the FPGA routing with simplified RC equivalent circuits and compute the delay of a routed connection as the Elmore delay of this equivalent circuit.

Tables 7.1 and 7.2 compare the delays computed by VPR, using these linearized RC models of transistors and buffers and the Elmore delay, to those computed by SPICE for various routing structures. Overall, the accuracy of our delay model is very good; the VPR delays are within 9% of those of SPICE for all but two cases. Those two cases consist of very short connections: a signal passing through a logic block output buffer to a wire segment of length 1 or length 4 with the routing switches loading this wire being pass transistors. The signal does not pass through any switch block routing switches. While the relative error in these two cases is 23.8% and 14.1% vs. SPICE, the delays are very small, so the absolute error in delay is only 112 ps and 89 ps, respectively. Consequently, these errors in delay will have little impact on the critical path delay reported by VPR.

TABLE 7.1 Delays computed by VPR vs. SPICE delays for buffered routing resources.

Wire Segment Length (in Logic Blocks)	VPR Delay (ns)	SPICE Delay (ns)	Difference
1	0.448	0.472	-5.1%
4	0.777	0.850	-8.6%
8	1.24	1.34	-7.5%
16	2.27	2.42	-6.2%
64	11.50	11.02	-7.9%

TABLE 7.2 Delays computed VPR vs. SPICE delays for pass-transistor-switched routing resources.

Wire segment length (in logic blocks)	Number of Wire Segments in Series	VPR Delay (ns)	SPICE Delay (ns)	Difference
1	1	0.359	0.471	-23.8%
4	1	0.543	0.632	-14.1%
8	1	0.816	0.876	-6.8%
16	1	1.46	1.47	-0.7%
1	4	0.883	0.92	-4.0%
4	4	2.32	2.14	+8.4%
8	4	4.70	4.33	+8.5%
16	4	11.04	10.61	+4.1%
1	16	7.03	6.48	+8.5%
4	16	24.06	22.10	+8.9%
8	16	54.06	52.03	+4.0%
16	16	139.6	141.5	-1.3%

7.2.4 Area Model

In this chapter we once again use the transistor-based area model described in Section 6.2.2. This model is built into VPR; once VPR has generated the routing resource

graph for the desired architecture, it traverses this graph, "builds" every structure required by the FPGA's routing, and determines the total number of minimum-width transistor areas required by the FPGA routing. To allow averaging of results from circuits of different sizes, we use a normalized area metric: the number of minimum-width transistor areas per tile (i.e. per logic block). All the results in this chapter give only the routing area of the FPGA, since the logic block is held constant throughout all the experiments. The logic block used in this chapter occupies 1678 minimum-width transistor areas, and hence the addition of 1678 to any of the routing area results presented in this chapter yields the total area per tile.

Importance of a Detailed Area Model

Most prior researchers have evaluated routing area either by counting the number of tracks per channel required to successfully route, or by counting the number of programmable switches in the routing. Counting the number of tracks required to route a circuit is not a good area metric for architecture studies (such as this study) in which the number of switches per track segment can vary, since the area required by each routing track is then variable. Counting the number of programmable switches in the routing is a better area metric, but is still not sufficiently accurate for our purposes. Modern FPGAs use three different types of programmable switch, and the different switches require considerably different layout areas. The connection blocks from routing tracks to logic block input pins are implemented with multiplexers; the connection blocks from logic block output pins to routing tracks, and some of the routing switches, are implemented via pass transistors; and some routing switches are tri-state buffers. Table 7.3 lists the area required by each of these switch types, including any area required by SRAM bits to control each switch. The area per switch varies by a *factor of 6.8* from the most area-efficient switch to the least area-efficient. Clearly, simply counting the number of programmable switches in a routing architecture does not provide a good area estimate.

TABLE 7.3 Comparison of programmable switch areas.

Switch Description	Minimum-Width Transistor Areas
Multiplexer (32 inputs)	2.88 per switch (92 / 32 inputs)
Multiplexer (4 inputs)	4.5 per switch (18 / 4 inputs)
Pass transistor (10x minimum drive)	11.5
Tri-state buffer (5x minimum drive)	19.7

7.2.5 Experimental Philosophy

The detailed routing architecture design space is huge, so we cannot simply explore the entire N-dimensional space of all possible detailed routing architectures. Instead, we perform a series of experiments in which we vary one or two architectural parameters, while fixing the other parameters to reasonable values. While varying a single parameter, we are optimizing along a line in the N-dimensional architecture space. Once we have found the best value of one parameter, we set it to this value, and vary a different parameter. Essentially we are exploring the design space in a manner similar to the way matrix solvers like the conjugate-gradient method explore the solution space of a matrix.

We begin with relatively simple architectures, and search for the best values of the parameters that are unlikely to interact strongly with other architectural parameters. We can then fix these parameters at their best values, and vary other architectural parameters to create increasingly complex architectures.

7.3 Single Wire Length Architectures

In this section we evaluate architectures in which every routing wire segment has the same length, and in which all the routing switches in switch blocks are tri-state buffers. Our goals in this section are to:

1. Determine the most appropriate switch block for segmented architectures,
2. Gain insight into the wire segment length that is most important in an FPGA, and
3. Find the best F_c value(s) for use with segmented architectures and the logic block used in this chapter (a cluster of size 4).

We ran the twenty largest MCNC benchmarks through the flow of Figure 7.1 and determined the routing area required and the critical path delay achieved by each circuit in each architecture of interest. Figure 7.2 plots the average (over the twenty circuits) of the critical path delay for each architecture, while Figure 7.3 plots the average routing area required in each architecture. The horizontal axis in both Figures 7.2 and 7.3 is the length of the routing wire segments. Recall that each architecture in Figures 7.2 and 7.3 potentially contains a different number of tracks per channel — each circuit is routed in an FPGA with 20% more routing tracks than the minimum the circuit needs to route in an FPGA with the given architecture.

In both Figures 7.2 and 7.3, the solid lines use the disjoint switch block topology [137] described in Section 4.2.3, while the dashed lines use the Wilton switch block

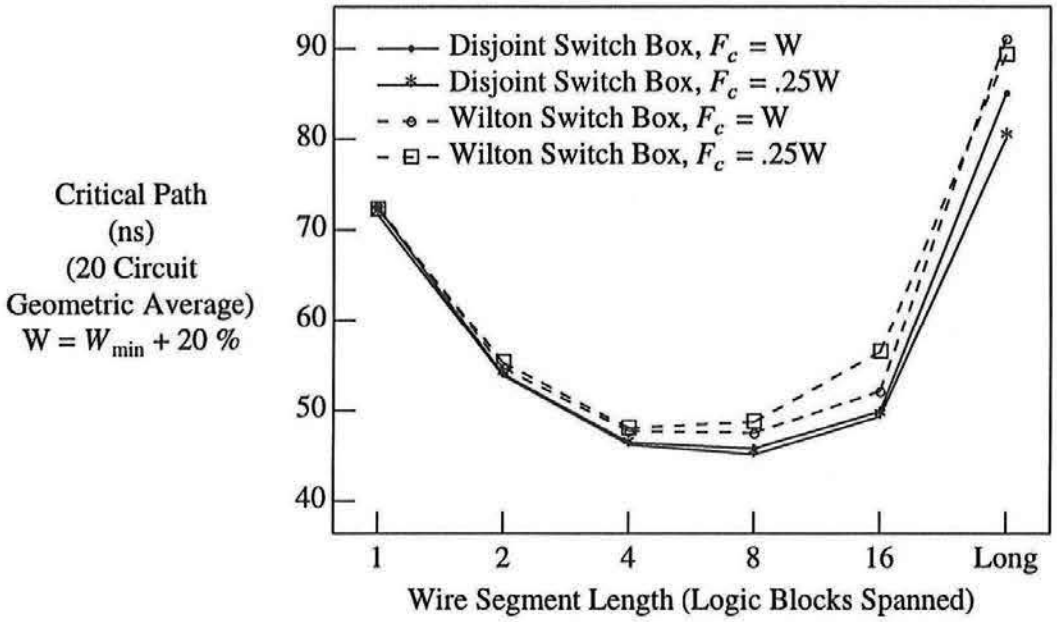


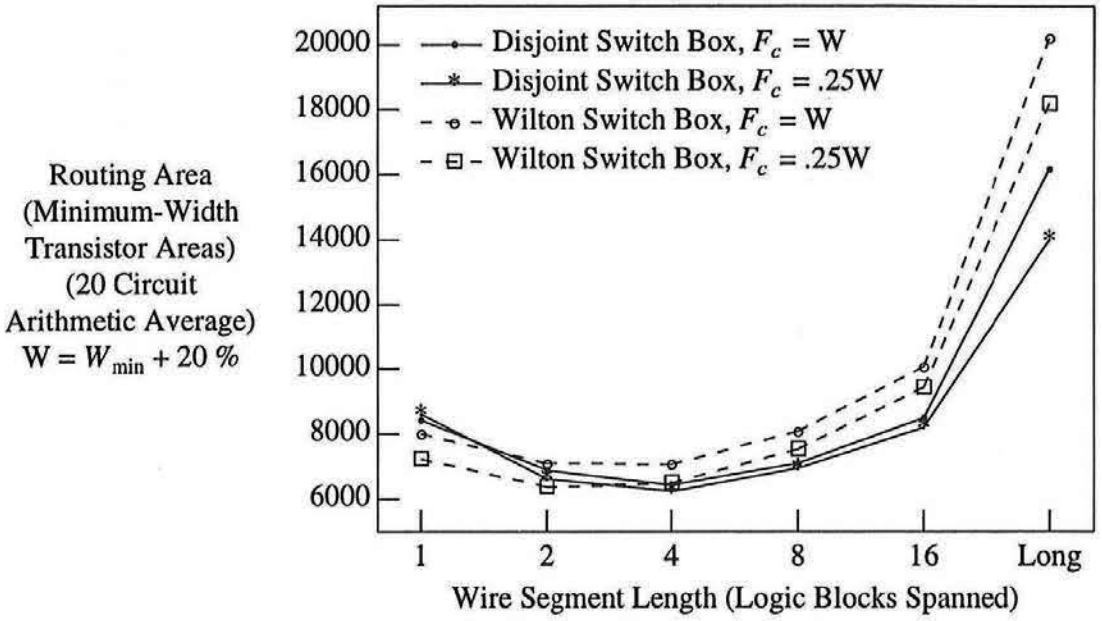
FIGURE 7.2 Circuit speed vs. switch block topology, F_c, and routing segment length.

topology [33]. Each curve plots the routing area or the critical path delay versus the length (in logic blocks) of the routing segments used in the FPGA. The "long" point on the horizontal axis indicates the performance of a routing architecture when all its wires are long lines that span the entire chip.

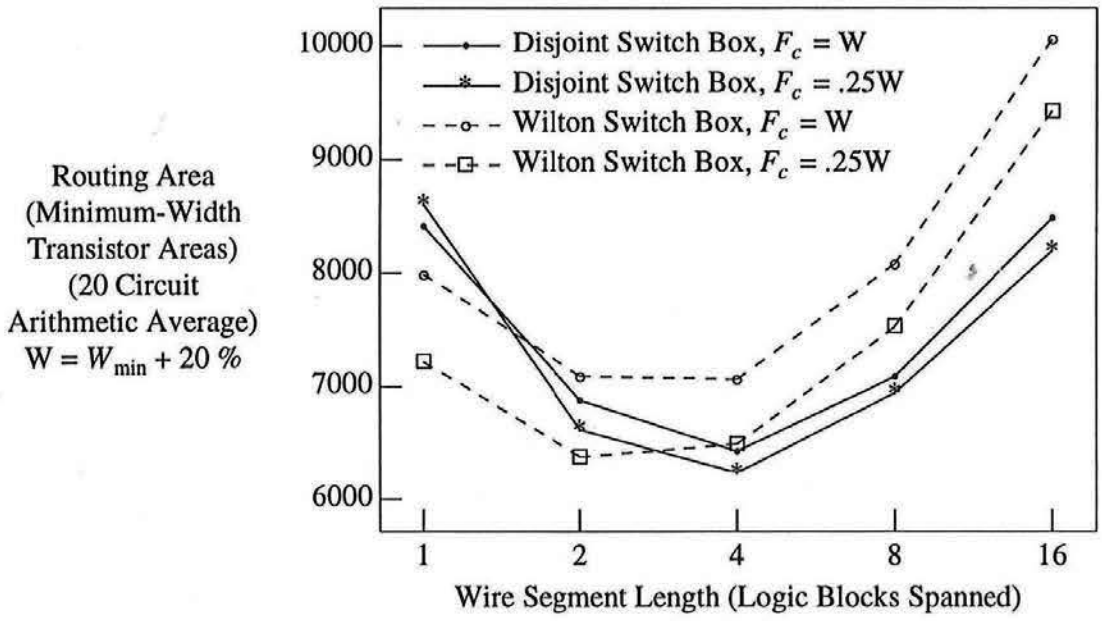
7.3.1 Switch Block Issues

The first feature of interest in Figures 7.2 and 7.3 is the performance of the two different switch blocks. Notice that the critical path delay achieved by both switch blocks is about the same for FPGAs composed of length 1 wires. However, as the length of the wire segments increases, the disjoint switch block leads to faster circuits than the Wilton switch block. Similarly, Figure 7.3 shows that the Wilton switch block leads to the best area-efficiency when all the routing wires span only one logic block. As the wire length increases to 2 logic blocks, however, the area required by the Wilton and disjoint switch blocks become similar and for wire lengths greater than 2 the disjoint switch block is more area-efficient than the Wilton switch block.

When a wire terminates at a switch block, both the disjoint and Wilton switch block topologies allow it to connect to three other wires, (i.e. $F_s = 3$). Hence both switch block topologies require the same number of switches in this case, as Figure 7.4(a) shows. Longer wires, however, do not always terminate at a switch block — they often pass through the switch block. Figure 7.4(b) shows this case. For the disjoint switch block, the six routing switches required by the two horizontal wire segments in



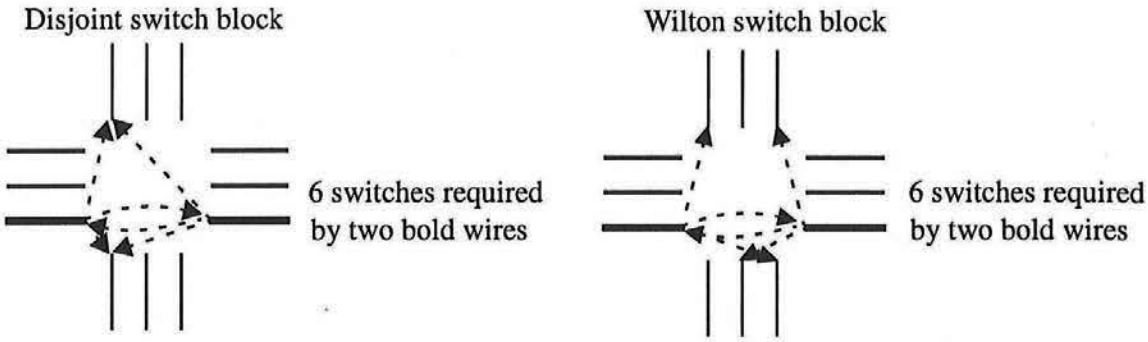
(a) Wire segment lengths from 1 logic block to the entire chip



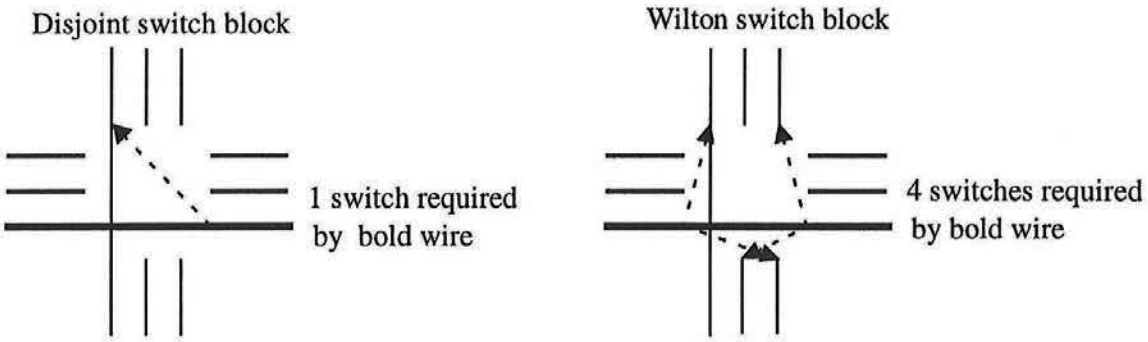
(b) Close-up of wire segment lengths from 1 to 16 logic blocks

FIGURE 7.3 Routing area vs. routing architecture: (a) all wire lengths; (b) close-up.

Figure 7.4(a) now all connect the same two pieces of metal together. Consequently we only need to implement *one* of these switches — the other five are redundant. In the Wilton switch block case, however, only two of the six switches that would be inserted if the wires ended at this point are redundant — a total of *four* routing



(a) Wires terminate at switch block — "end" switch point



(b) Wires continue through switch block — "internal" switch point

FIGURE 7.4 Switch block examples at (a) end and (b) interior of wires with length greater than 1.

switches are still required for this wire segment at this switch block. So at "internal" switch blocks, where a wire segment passes through the switch block without terminating, the disjoint switch block has an $F_{s,internal}$ of one, while the Wilton switch block has an $F_{s,internal}$ of up to four. This is a key point — switch blocks that use the same number of switches for FPGAs in which all wires are of length one can use *different* numbers of switches in FPGAs with longer wires. As well, it is generally not possible to describe switch block topologies with a single F_s number when some routing wires span more than one logic block.

The reason for the behaviour seen in Figures 7.2 and 7.3 is now clear. If all the wires in an FPGA have length one, the Wilton and disjoint switch blocks use the same number of switches per wire. The Wilton switch block results in better routability [33], however, so it requires fewer wires per channel, and hence less area. As the wire length increases, however, we have more and more wires passing through switch blocks without terminating, and hence more and more "internal" switch points. Since the Wilton switch block uses more switches at these points than the disjoint switch block, it loads the wires more heavily, degrading speed. As well, the reduction in the

number of wires per channel required to successfully route a circuit is not large enough to outweigh the area of these extra switches, so the Wilton topology's area-efficiency is worse than that of a disjoint topology for FPGAs with longer wires.

These switch block results are interesting because prior researchers have focused on determining the best switch block for architectures that contain only length 1 wires. Clearly the best switch block for such an architecture is *not* necessarily the best switch block for architectures containing longer wires, so research into good switch blocks for FPGAs with longer wires is a fertile area for research. Throughout the remainder of this chapter we will use the disjoint switch block. This switch block topology is almost certainly suboptimal, however. The fact that this switch block only allows wires in track i to connect to other track i wires reduces routability somewhat, and it means that wires of differing lengths can never be connected together. The routing of some nets would likely benefit from the ability to connect wires of differing lengths. For example, one intuitively expects that a good way to route a high-fanout net is to use long wire segments to create a "backbone" that spans the FPGA, and then use shorter wires to reach any sinks that are not immediately adjacent to the backbone. VPR can route FPGAs with arbitrary switch blocks, so a useful future research project would be to employ our tools in a search for better switch blocks for FPGAs that contain longer wires.

7.3.2 Best Single Wire Length

From Figure 7.2, one can see that the fastest FPGA which uses only one length of wire uses wires of length 4 or length 8, regardless of the switch block and F_c value used. Shorter wires lead to poor speed because long connections must pass through too many buffers. Very long wires degrade speed in two ways. First, short connections are forced to use long wires, which are slower than short wires due to their larger capacitance. Second, even connections that travel the entire length of a long wire become slow when the wire is too long because the metal resistance of the wire becomes large, and eventually reduces speed below that of a larger number of short wires connected by buffers.

Figure 7.5 quantifies the relative speeds of short and long wires, and provides insight into why FPGAs using length 4 or length 8 wires have the best performance. Figure 7.5 shows the delay *per logic block* spanned by a connection as a function of the length of wire used, when the routing switches between wires are tri-state buffers. Increasing the wire length from 1 to 2 logic blocks increases the speed of long connections by 61% by reducing the number of routing switches in series by a factor of 2. As wire length increases past 4 logic blocks we clearly have diminishing speed returns. Note that a wire of length 32 (for metal layer 2) or of length 64 (for metal layer 3) is *slower,* even for long connections that span the entire wire, than the series

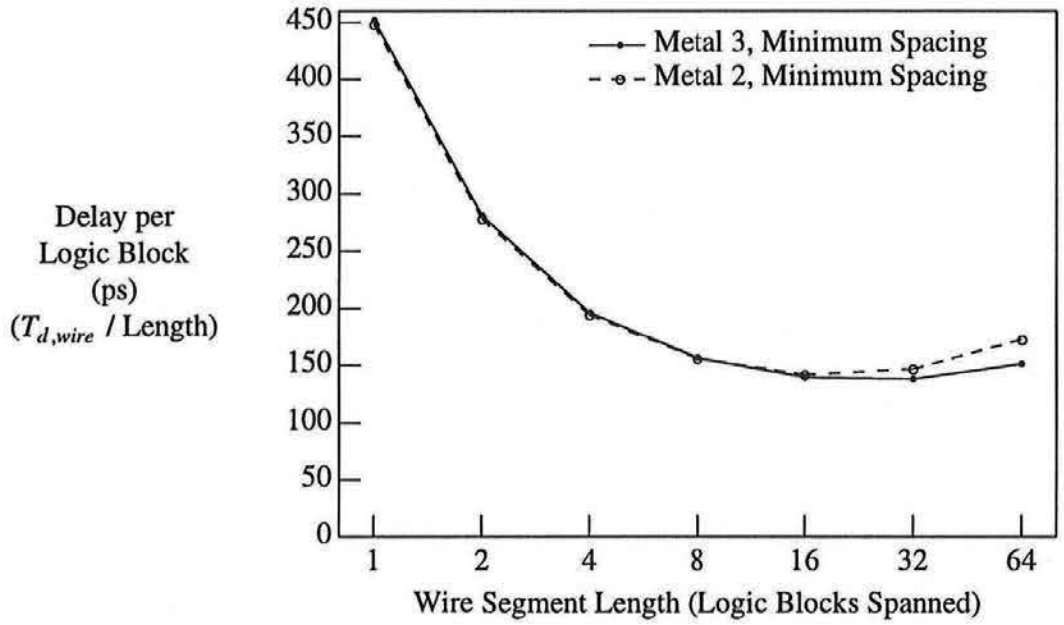


FIGURE 7.5 Delay per logic block spanned by a connection vs. wire segment length.

connection of several shorter wires. Since metal 3 has a lower resistance, wires can be made somewhat longer (64 logic blocks rather than 32) before the metal resistance slows them down more than extra routing switches would. Since very long wires are also less flexible than short wires in terms of the connections for which they can be used, it is clear that very long wires are not very useful for FPGAs in modern IC processes.

Figure 7.3 shows that wires of length 2 or length 4 lead to the most area-efficient FPGA architecture. Length 2 wires are the most area-efficient if the Wilton switch block topology is used, while length 4 wires are the most area-efficient if the disjoint switch block topology is used. As we increase the length of the routing wires two competing factors determine the resulting architecture's area-efficiency. First, longer wires are less "flexible"; they cannot be split in the middle, so short connections will waste part of a wire segment. This means the number of tracks per channel required to successfully route a circuit increases as the wire segment length increases. On the other hand, longer wires pass through more switch blocks before terminating, so the fraction of "internal" switch points in switch blocks increases. As Figure 7.4 shows, these internal switch points require fewer programmable switches, resulting in decreased area. When the disjoint switch block is employed, each internal switch point requires only one programmable switch; with a Wilton switch block, internal switch points require up to 4 programmable switches. Consequently, longer wire segments reduce the number of switches per wire more dramatically with a disjoint switch block than with a Wilton switch block.

Length 4 wires achieve the best combination of low delay and high area-efficiency. An architecture using all length 8 wires can achieve slightly (~2%) better speed, but requires 11% more area. While real FPGA architectures can of course use more than one wire length, this result is evocative. It leads one to expect that the best FPGA architectures will either include significant numbers of length 4 or length 8 wires, or will include some wires shorter and some wires longer than length 4.

7.3.3 Amount of Connectivity Between Logic Blocks and Channels

Figure 7.2 shows that F_c has little effect on circuit speed. Figure 7.3 shows that an F_c value of 0.25·W is superior in terms of area-efficiency to F_c = W for any segment length if the Wilton switch block topology is used. If the disjoint switch block topology is used, F_c = 0.25·W is superior to F_c = W for all wire segment lengths except length = 1.

Figure 7.6 shows how routing area varies with F_c when all routing wires span 4 logic blocks (we use length 4 wires since they were found to be the most area-efficient in the previous section). For the Wilton switch block topology, the best area-efficiency occurs at F_c = 0.25·W. When the disjoint switch block topology is used, however, the best value of F_c is 0.5·W. Since the disjoint switch block is less routable than the Wilton switch block, a higher value of connection block flexibility is needed for the best area-efficiency.

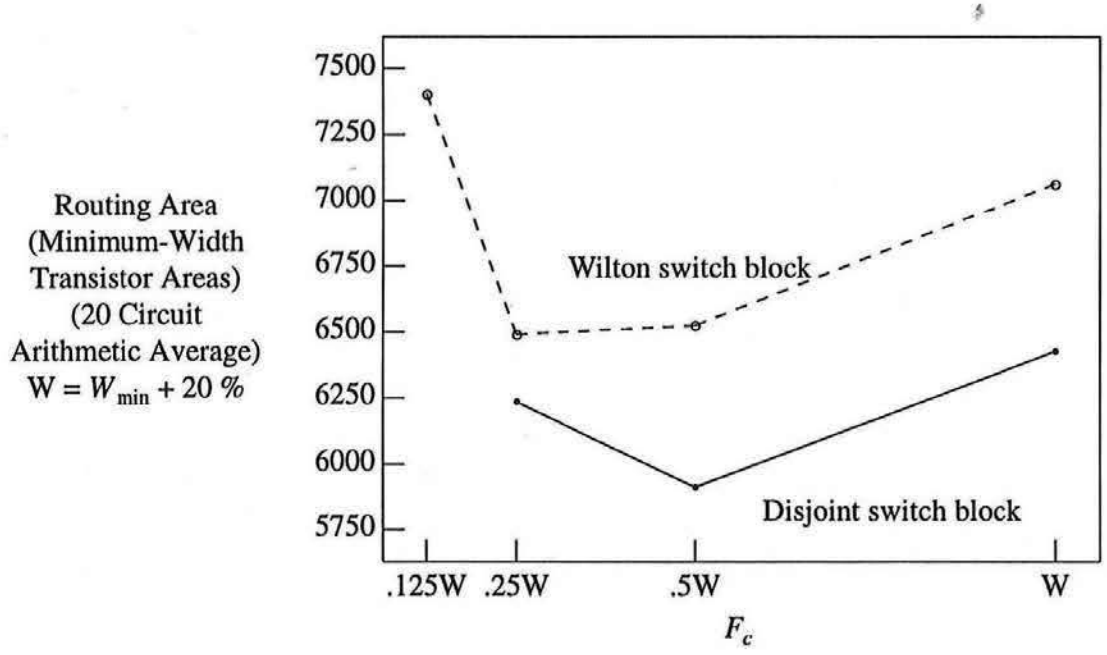


FIGURE 7.6 Routing area vs. F_c for a routing architecture with all length 4 lines.

With the disjoint switch block topology some additional routing area can be saved by using a lower value F_c value for logic block output pins ($F_{c,output}$) than for input pins ($F_{c,input}$). Recall that the connection block from a logic block output pin to routing tracks is constructed via $F_{c,output}$ pass transistors controlled by $F_{c,output}$ SRAM cells. The connection block from routing tracks to a logic block input pin, on the other hand, is constructed via a multiplexer with $F_{c,input}$ inputs. The area per switch is therefore higher for the logic block output pin switches than for the switches to logic block input pins, as Table 7.3 shows. Consequently, making $F_{c,output}$ lower than $F_{c,input}$ makes intuitive sense. We have found that, with the disjoint switch block, the best value of $F_{c,output}$ is 0.25·W, while the best value of $F_{c,input}$ is 0.5·W. This lower value of $F_{c,output}$ saves some routing area (2% - 5%, depending on the exact routing architecture) versus using an $F_{c,output}$ of 0.5·W. An $F_{c,output}$ value of 0.25·W ensures that each routing track can be driven by one output on each logic block (since our logic block has four logically-equivalent outputs).

7.4 Two Types of Wire Segment Architectures

In this section we examine somewhat more complex architectures: those that contain two *types* of wire segments. Two wires are of different types if their lengths are different, or if they use different types of routing switches to connect to other wires (e.g. pass transistors vs. tri-state buffers). All the experiments in this section use the disjoint switch block topology and an F_c value of 0.5·W, since the previous section showed these are good choices for a wide range of architectures.

7.4.1 Tri-State Buffer Routing Switches Only

We investigated a large number of architectures that contained two different lengths of routing wires, and in which all the routing switches in each switch block were tri-state buffers. We found that we could achieve only small improvements compared to the best single wire length architecture (length = 4). Table 7.4 compares the performance of the two best architectures investigated in this section to the best single wire length architecture. Both these architectures are fairly similar to an architecture in which all wires have length 4 — one has 25% length 2 wires and 75% length 8 wires, while the other has 75% length 4 wires and 25% length 8 wires. Three columns in Table 7.4 compare the speeds of these architectures to that of a length 4 architecture under three different assumptions of the metal layer and spacing used for routing wire segments. The average speedup vs. a length 4 architecture is only 5.3% for the first architecture, and 5.7% for the second. Both of these architectures are slightly *less*

dense than an architecture that contains only length 4 wires. Clearly length 4 wires provide an efficient way to make both short and long connections!

TABLE 7.4 Best buffered, two different wire length architectures vs. best single wire length architecture. Average over 20 benchmark circuits.

Architecture	Critical Path Delay (ns)			Routing Area (Minimum-Width Transistor Areas)
	Metal 2, Min. Spacing	Metal 3, Min. Spacing	Metal 3, Wide Spacing	
All length 4, buffered	45.26	45.62	39.81	5901
25% length 2, 75% length 8	43.45	43.74	37.08	6034
75% length 4, 25% length 8	43.18	43.44	37.06	5948

7.4.2 Length 4 Buffered Wires Plus Pass-Transistor-Switched Wires

Since a routing architecture composed solely of length 4 wires that use tri-state buffers as their routing switches performs so well, in this section we investigate architectures in which some routing tracks contain wires of this type. The other routing tracks contain wires that connect to each other with pass transistor routing switches. We will investigate different lengths of these "pass-transistor-switched" wires, and different proportions of the two types of wires.

Figure 7.7 shows the speed achieved by FPGA architectures in which 50% of the routing tracks use length 4 wires connected by buffered switches, and the other 50% consist of some other length of wires connected with pass transistors. The horizontal axis in Figure 7.7 is the length of the pass-transistor wires used. The three different curves in Figure 7.7 assume three different metal layer / spacing options for routing wires; clearly the metal width and spacing used does not significantly change the shape of the curve.

Figure 7.8 provides the area results for these architectures. The best area-efficiency occurs when the pass-transistor-switched wires are length 2, but length 4 and length 8 wires also have reasonable area-efficiency, and Figure 7.7 showed they lead to superior speed.

Figures 7.9 and 7.10 investigate the performance of length 4 buffered wires combined with either length 1, 2, 4, or 8 pass-transistor-switched wires in different proportions, (i.e. not just 50 / 50). The horizontal axis in these figures is the fraction of routing tracks composed of the pass-transistor-switched wires; the remainder of the routing

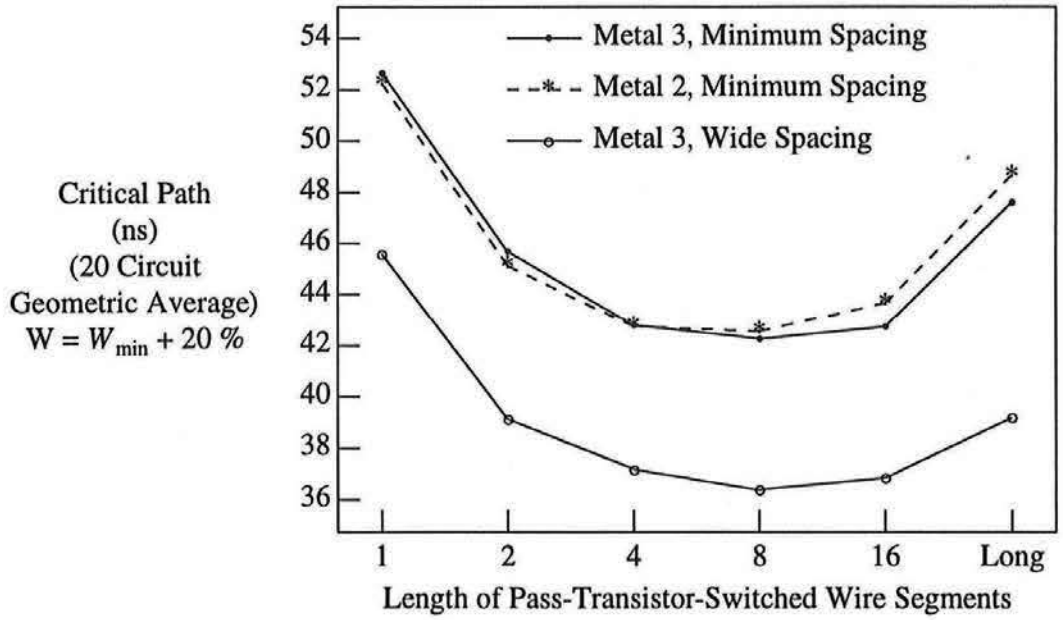


FIGURE 7.7 Speed of architectures with 50% length 4 buffered and 50% pass-transistor-switched wires.

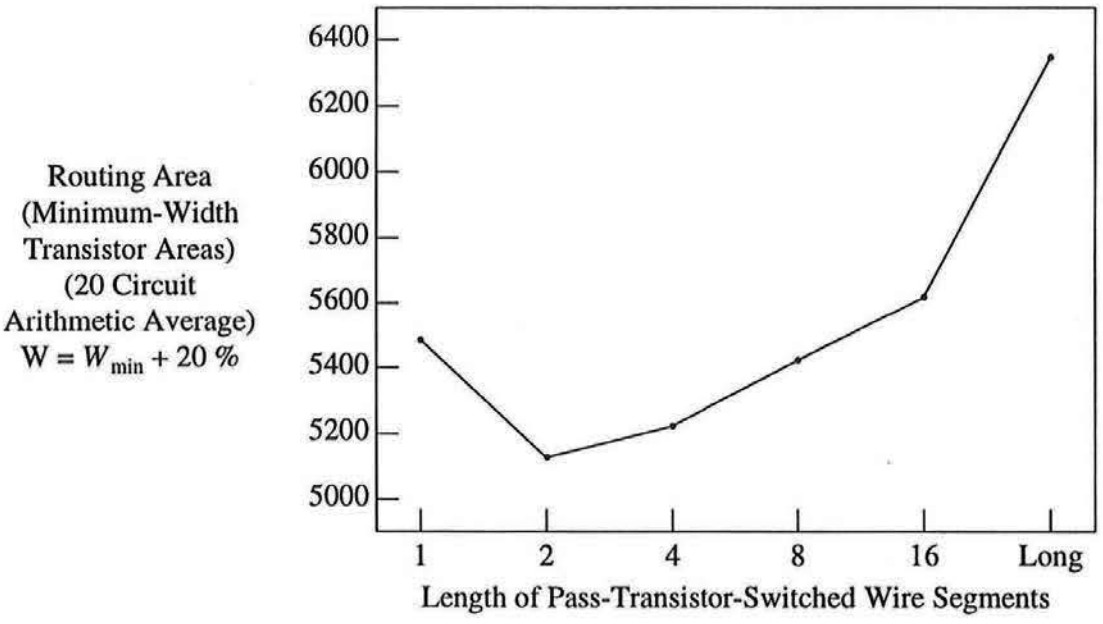
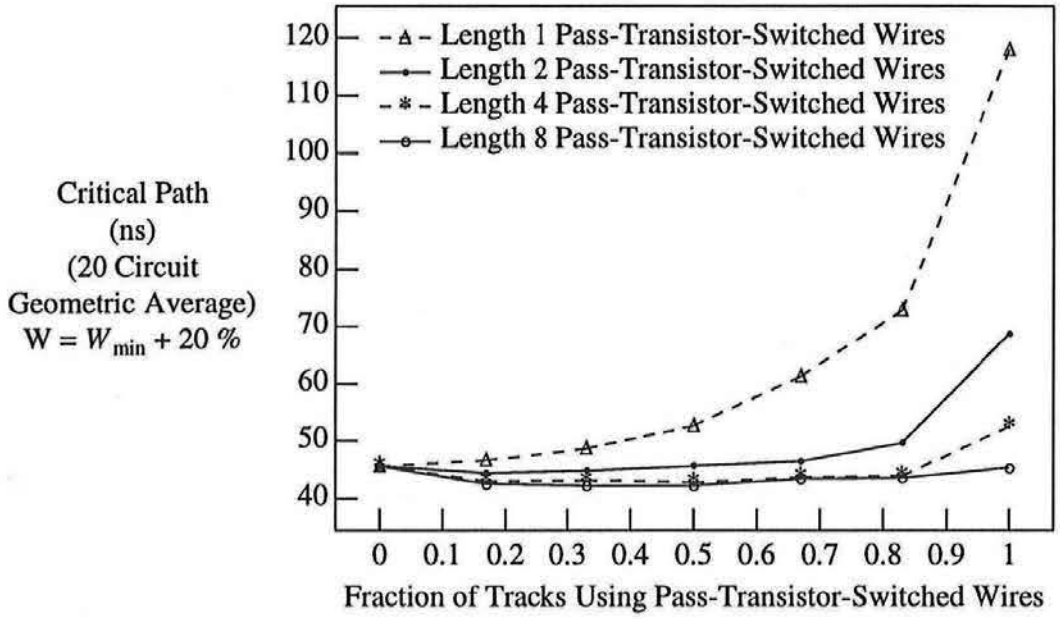
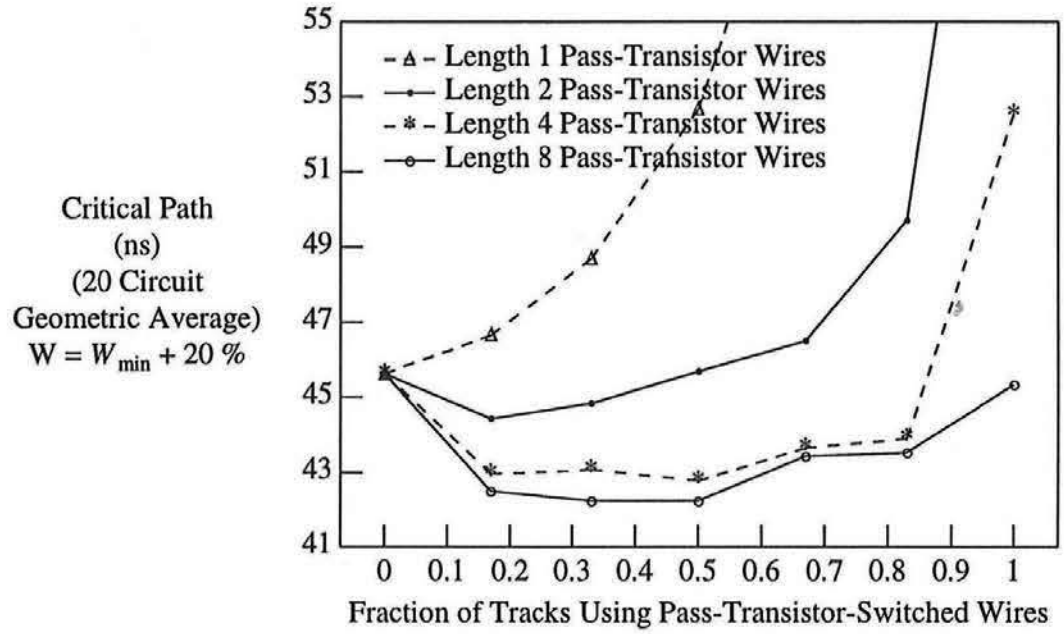


FIGURE 7.8 Area of architectures with 50% length 4 buffered and 50% pass-transistor-switched wires.



(a) All architectures



(b) Close-up of architectures with good speed

FIGURE 7.9 Speed of FPGAs with a mix of length 4 buffered wires and pass-transistor-switched wires.

tracks are composed of length 4 buffered wires. The "0" point on the horizontal axis corresponds to an architecture composed solely of length 4 buffered wires, while the "1" point corresponds to architectures composed solely of wires that connect to each other via pass transistors. Figure 7.9 shows the speed of the various architectures,

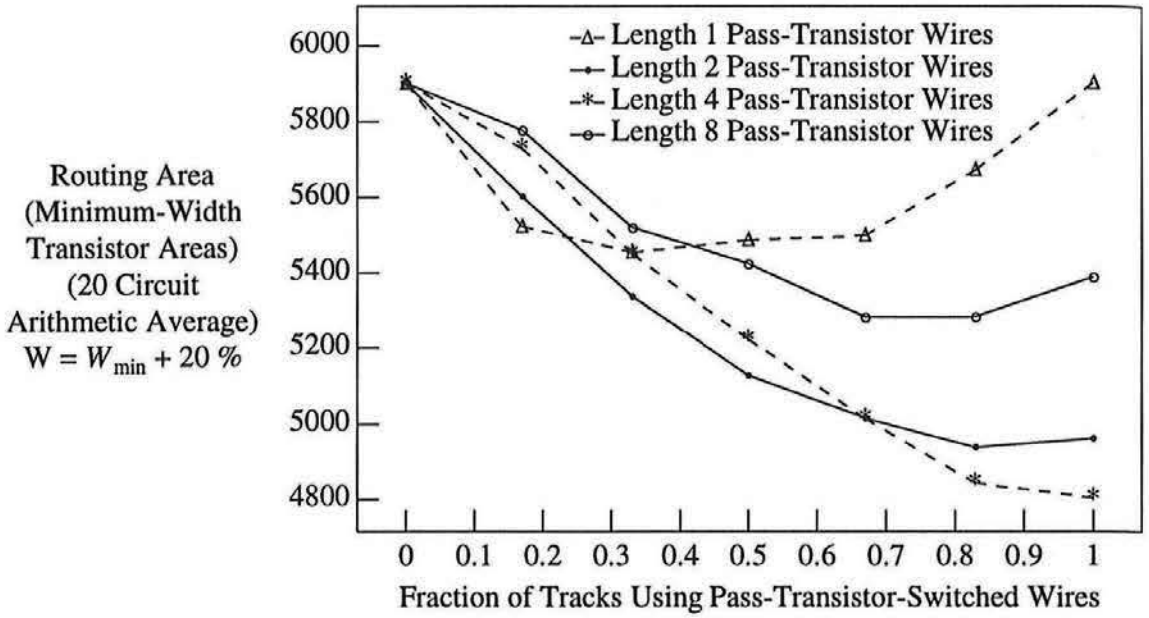


FIGURE 7.10 Area of FPGAs with a mix of length 4 buffered wires and pass-transistor-switched wires.

assuming routing wires are implemented in metal 3 with the minimum metal spacing, while Figure 7.10 shows their area. Clearly, adding length 1 wires to an architecture is not a good idea. If 33% of the routing tracks are length 1 pass-transistor-switched wires, then area-efficiency improves by 8% (vs. all tracks being length 4 buffered wires), but speed degrades by 7%. Larger fractions of length 1 wires degrade both area and speed — these wires are simply too short to be of much use. Many commercial architectures make heavy use of length 1 wires [4, 139], but our results suggest they could improve both their speed and area-efficiency by using longer wires instead. Note that the most widely studied architecture in academic research, an architecture composed entirely of length 1 wires connected by pass transistors, has extremely poor speed — it is 2.8 times slower than the fastest architecture in Figure 7.9.

Adding longer pass-transistor-switched wires to an architecture yields better results. Figure 7.9 shows that making between 17% and 83% of the routing tracks pass-transistor-switched wires of length 4 or 8 increases the FPGA speed. Pass transistors do not have the intrinsic delay of the multi-stage buffers used in buffered routing switches, and they have higher drive strength than a tri-state buffer for the same area, since they use only one transistor, rather than several. On the other hand, the delay through a series chain of pass transistors grows quadratically with the number of pass transistors. The net effect is that pass transistor switches are faster than buffers for connections that pass through a few series switches, but buffers are faster for connections that pass through many series switches. If long wires are used, fewer series routing switches are needed for long connections, making pass transistor switches

more competitive with buffers for longer connections. Consequently a mix of moderate length buffered and moderate length pass-transistor wires leads to better speed than using all buffered routing switches or all pass transistor routing switches.

Figure 7.10 shows that increasing the fraction of routing tracks using length 2, 4 or 8 pass-transistor wires improves the FPGA area-efficiency until this fraction reaches approximately 83%; after that area-efficiency degrades (or levels off, for length 4 wires). A pass transistor switch requires less area than a tri-state buffer, and since pass transistors are bidirectional, one pass transistor can replace two tri-state buffers in the routing. On the other hand, pass transistor switches are not well-suited to routing high-fanout nets, as described in Section 4.4.4. To maintain reasonable speed, a high-fanout net routed using pass-transistor switches tends to use a "star" topology. This requires more wiring, and hence more routing tracks, and hence more area. Making all routing switches pass transistors forces even high-fanout nets to be routed using pass transistors, degrading area-efficiency.

TABLE 7.5 Best architectures combining length 4 buffered and pass-transistor-switched wires.

Wire Distribution of Architecture	Delay (ns)	Speedup vs. Length 4 Buffered Architecture	Area (Minimum-Width Transistor Areas)	Area vs. Length 4 Buffered Architecture
100% Length 4 Buffered	45.62	—	5901	—
50% Length 4 Buffered, 50% Length 4 Pass Transistor	42.78	+6.6%	5223	-11.5%
17% Length 4 Buffered, 83% Length 4 Pass Transistor	43.90	+3.9%	4843	-17.9%
50% Length 4 Buffered, 50% Length 8 Pass Transistor	42.25	+8.0%	5424	-8.1%
17% Length 4 Buffered, 83% Length 8 Pass Transistor	43.54	+4.8%	5285	-10.4%

Considering both area and speed, the best architectures use 50% - 83% pass-transistor switched wires, with these wires having a length of 4 or 8. Table 7.5 summarizes the performance of several of the best architectures found in this section, and lists performance of an FPGA using all length 4 buffered wires for comparison. The area improvement due to a mix of pass transistors and buffers ranges from 8% to 18%, while the speed improvement ranges from 4% to 8%. The architectures with 83% pass-transistor switched routing tracks have better area-efficiency, at the cost of

slightly reduced speed, compared to FPGAs with only 50% pass-transistor switched routing tracks.

A major conclusion of this section is simply that the best routing architecture contains a mix of pass transistors and tri-state buffers. This fact is not widely known. Virtually all prior academic research has focused on FPGAs that contain only pass transistors, while a new FPGA start-up company has made the fact that their FPGAs contain no pass transistors (all routing switches are tri-state buffers) a marketing feature [146].

7.4.3 Length 8 Buffered Wires Plus Pass-Transistor-Switched Wires

In Section 7.3.2 we found that an FPGA composed entirely of length 8 wires interconnected by tri-state buffers performed almost as well as an FPGA that used only length 4 wires. Consequently, in this section we investigate routing architectures in which some routing tracks are composed of length 8 buffered wires, while the other routing tracks are composed of wires connected by pass transistors (and which may have a length different than 8). Our goals are to find the best length of pass-transistor switched wire to combine with length 8 buffered wires, and to find the best proportion of pass-transistor-switched routing tracks to length 8 buffered routing tracks.

The architectural conclusions of this section are very similar to those of the previous section:

1. The best combination of area and delay results when the pass-transistor switched wires are of length 4 or 8.
2. The best architectures contain from 50% to 83% pass-transistor-switched routing tracks, with the 50% pass-transistor architectures giving the best speed, and the 83% pass-transistor architectures yielding the best area-efficiency.

Table 7.6 summarizes the performance of the best architectures that combine length 8 buffered wires with some other length of pass-transistor switched wires. The performance of an architecture using all length 8 buffered wires is also listed for comparison. Comparing these results to those of Section 7.4.2 shows that using length 8 buffered wires instead of length 4 buffered wires has slightly increased the FPGA speed, at the cost of slightly decreased area-efficiency.

7.4.4 Length 4 Pass-Transistor-Switched Wires Plus Buffered Wires

Recall from Section 7.2.5 that we are exploring the highly complex detailed routing architecture space by fixing most of the architectural parameters and varying only one

TABLE 7.6 Best architectures combining length 8 buffered and pass-transistor-switched wires.

Wire Distribution of Architecture	Delay (ns)	Speedup vs. Length 8 Buffered Architecture	Area (Minimum-Width Transistor Areas)	Area vs. Length 8 Buffered Architecture
100% Length 8 Buffered	45.03	—	6339	—
50% Length 8 Buffered, 50% Length 4 Pass Transistor	41.29	+9.1%	5227	-17.5%
17% Length 8 Buffered, 83% Length 4 Pass Transistor	43.69	+3.1%	4869	-23.2%
50% Length 8 Buffered, 50% Length 8 Pass Transistor	41.56	+8.3%	5732	-9.6%
17% Length 8 Buffered, 83% Length 8 Pass Transistor	43.15	+4.4%	5447	-14.1%

or two in a set of experiments. The danger in such an exploration approach is that we may miss some important architectures by fixing some parameters at values that initial experiments led us to believe were reasonable, but which were suboptimal for the more complex architectures studied later.

To guard against such occurrences, we often investigated a certain portion of the architecture space from more than one perspective. For example, in this section we again investigate FPGAs in which some of the routing tracks consist of wires that connect via pass-transistor switches, while the other routing tracks connect via tri-state buffers. In this section, however, we assume that the pass-transistor-switched wires are of length 4, and search for the length of buffered wiring segment that best complements these wires and leads to the best FPGA.

Figure 7.11 shows how circuit delay varies with the length of the buffered wiring segments, while Figure 7.12 illustrates the variation of FPGA routing area with the length of the buffered routing segments. In both Figures 7.11 and 7.12, 50% of the routing tracks are composed of length 4 wires that connect via pass-transistor switches, while the other 50% of the routing tracks use some length of buffered wire segment. Figure 7.11 shows that the best delay occurs when the buffered wire segments have length 8, although length 4 and length 16 wire segments lead to FPGAs that are only slightly slower. Figure 7.12 indicates that the best area-efficiency occurs when the buffered wire segments have length 4 or length 8.

Considering both speed and area-efficiency, the best architecture studied in this section combines length 4 pass-transistor wires with length 8 buffered wires, and the second best architecture uses length 4 buffered wires instead of length 8 buffered wires. The fact that length 8 and length 4 buffered wires were again found to be the best

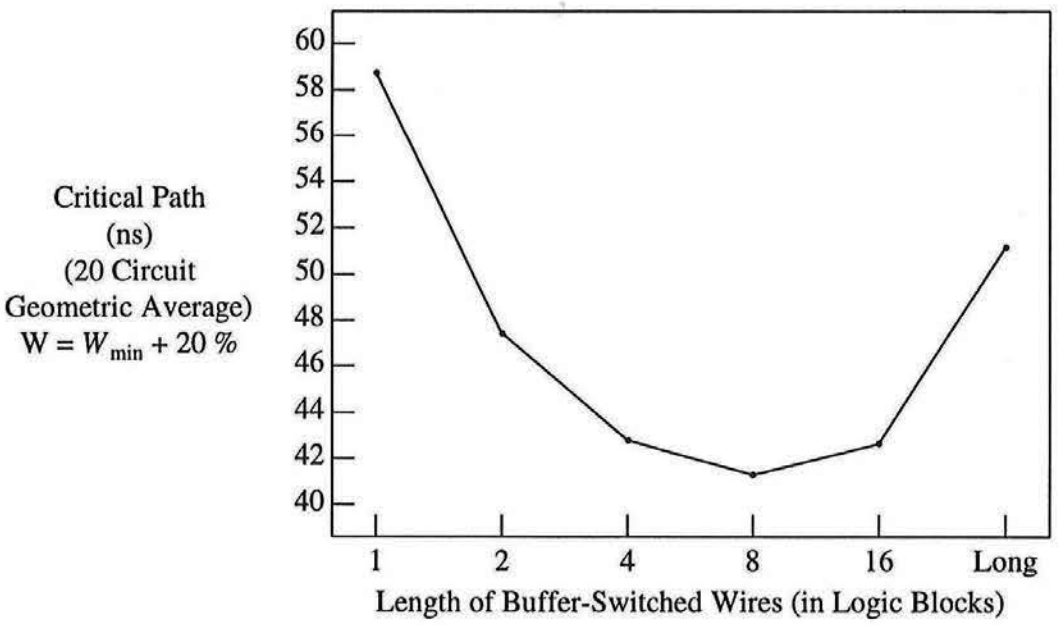


FIGURE 7.11 Speed of FPGAs using 50% length 4 pass-transistor wires and 50% buffered wires.

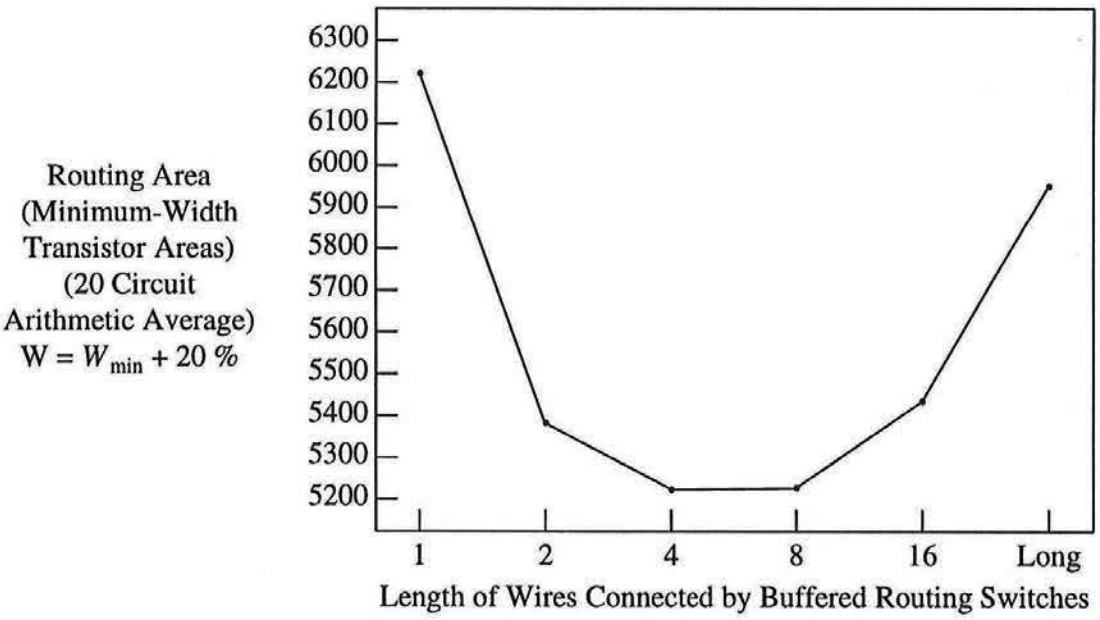


FIGURE 7.12 Area of FPGAs using 50% length 4 pass-transistor wires and 50% buffered wires.

increases our confidence that we fully explored the most promising two-wire-type architectures in Sections 7.4.2 and 7.4.3.

7.5 Internal Population

In this section we investigate the internal population question, which was described in Section 4.2.1. Recall that the connection-block internal population of a wire specifies how many of the internal points along the length of a wire that spans multiple logic blocks are accessible by switches to logic blocks. Similarly, switch-block internal population specifies how many of the internal points along a wire can connect to other wires via routing switches. In this section we determine what fraction of an FPGA's routing tracks should be composed of internally-depopulated wires, and exactly how depopulated these wires should be.

In all the experiments of this section we assume that routing wires are laid out in metal 3 with minimum spacing. $F_{c,output}$ is always 0.25·W, $F_{c,input}$ is always 0.5·W, and the switch block topology is disjoint.

7.5.1 All Length 4 Buffered Wires

In Section 7.3.2 we found that the best FPGA architecture that employs only one length of wire connected by buffers was an FPGA with length 4 wires. Consequently, in this section we investigate architectures that use all length 4 wires connected by tri-state buffers with varying amounts of connection-block and switch-block internal population.

We first performed a set of experiments in which all the wires in the architecture had the same amount of internal population. We found that if the wires were more than slightly depopulated, some circuits became completely unroutable (would not route at *any* channel width). More specifically, if the connection-block population is less than 75% (a wire connects to fewer than 3 out of the 4 logic blocks it spans) some circuits become unroutable. Similarly, if the switch-block population is less than 80% (a wire connects to other wires at fewer than 4 out of the 5 switch blocks it spans) some circuits will not route. Figure 7.13 shows why reducing the connection-block population to 50% makes some circuits unroutable. In Figure 7.13 the logic block has only one output, which is located on the bottom side of the logic block. Notice that when all the routing wires are length 4 and have a connection-block population of 50%, there is *no possible way* to connect many logic blocks. For example, logic block *A* in Figure 7.13 cannot connect to either of the shaded logic blocks. Similar, but more

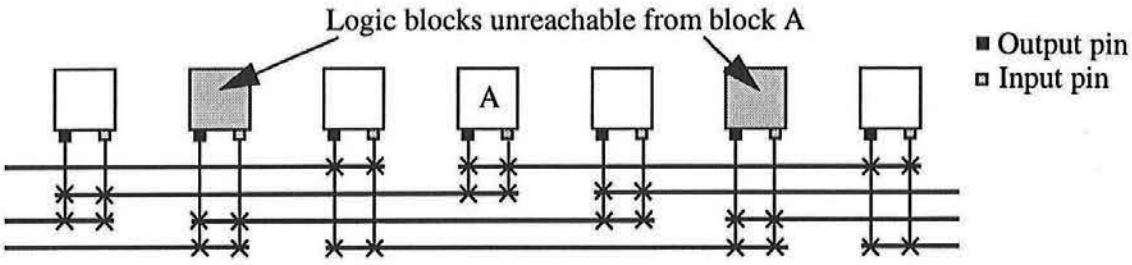


FIGURE 7.13 An FPGA with all length 4 wires and 50% connection-block population.

subtle effects render many circuits unroutable in FPGAs in which all the wires have less than 80% switch-block population. Clearly then one should not depopulate all the wiring tracks in an FPGA — some wires should remain fully populated to guarantee it is possible to connect every pair of logic blocks in the FPGA.[1]

Figures 7.14 and 7.15 show how circuit speed and routing area, respectively, are affected as the internal switch population of 50% of the routing wires in an FPGA is

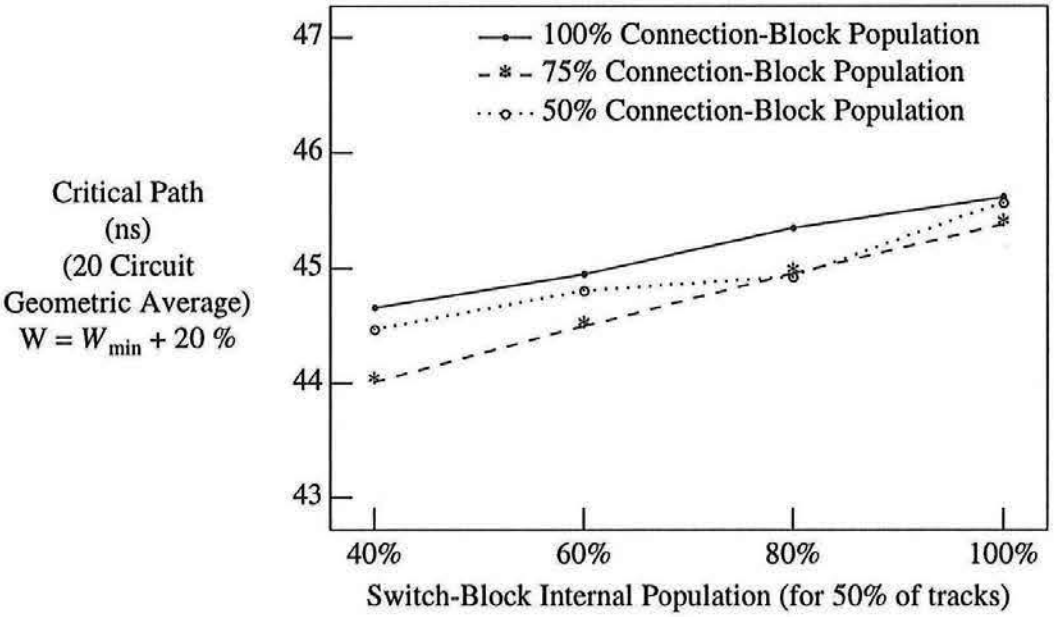


FIGURE 7.14 Speed of architectures with 50% of routing tracks depopulated in some manner.

1. A useful direction for future research would be to explore wire segments with differing amounts of internal connection-block population for logic block inputs and outputs. Switches to logic block output pins require the most area, so depopulating these switches yields the largest area gains. Keeping logic block input pin switches fully populated for a wire guarantees that a logic block can always reach any other logic block even if all wires are internally output connection-block depopulated.

varied. The other 50% of the routing wires are always fully populated. Each line in Figures 7.14 and 7.15 corresponds to a different amount of connection-block population for 50% of the routing wires, while the switch-block internal population for these wires varies along the horizontal axis. Figure 7.14 indicates that reducing the internal population of routing wires leads to a small (2% - 3%) speed gain. While reducing the internal population of a wire reduces its capacitive loading somewhat, most of the capacitance loading a wire is the metal capacitance of the wire itself, and this limits the achievable speedup.[1] As well, as the internal population of a wire segment is reduced, some nets may have to use more circuitous routes to connect their terminals because of the reduced flexibility of the routing fabric. This tends to decrease speed, and offsets some of the speed gain due to reduced capacitive loading.

Figure 7.15 details the effect of internal population on routing area. As the internal population of wires is reduced, the number of switches per wire drops, saving routing area. On the other hand, more tracks per channel are required to successfully route circuits because of the reduced flexibility of the routing, and this increases routing area. The best routing area is achieved when the internal population of the routing wires best balances these two factors. Figure 7.15 shows that reducing the connec-

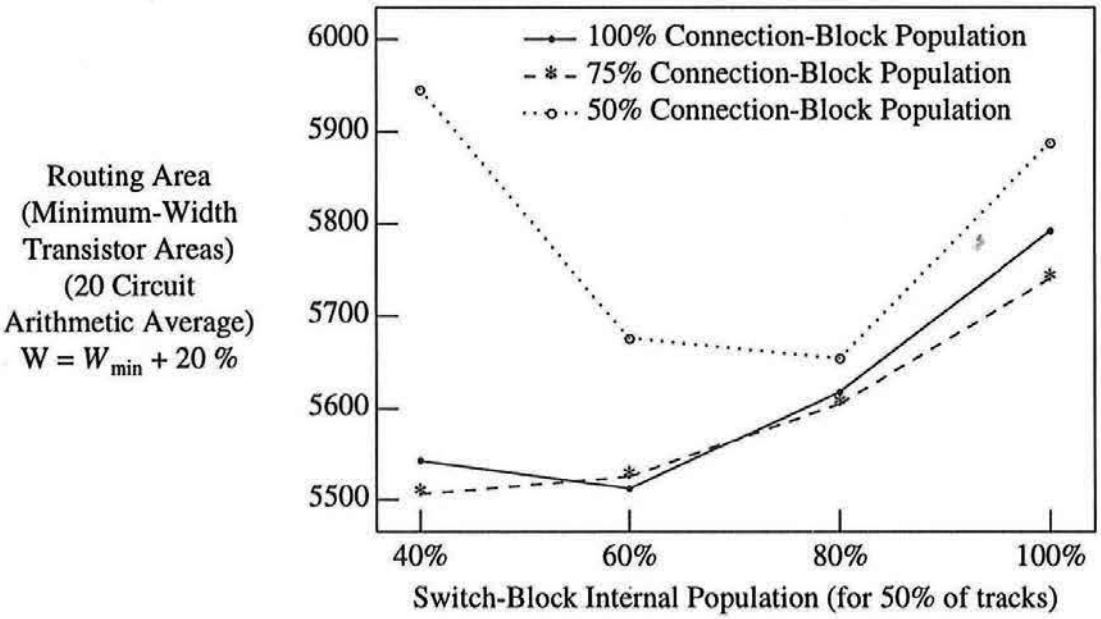


FIGURE 7.15 Area of architectures with 50% of routing tracks depopulated in some manner.

1. Larger speed gains are possible if one sizes up the routing transistors connecting to the internally-depopulated wire segments, as this reduces the portion of the delay due to metal capacitance. This requires additional area, but may be desirable when an FPGA architect is more concerned with speed than area.

tion-block population from 100% to 75% has only a marginal impact on area-efficiency, and reducing it further, to 50%, degrades area-efficiency. Reducing switch-block population is more effective — there is a clear area win as the switch-block internal population is reduced from 100% to 60%. Further reductions to a switch-block population of 40% provide little or no area gain. The three best architectures are those in which 50% of the routing tracks have:

- 100% connection-block population, and 60% switch-block population, or
- 75% connection-block population and 60% switch-block population, or
- 75% connection-block population and 40% switch-block population.

Figure 7.16 shows the average number of tracks per channel required to successfully route the benchmark circuits (W_{min}) for the various architectures. Notice that W_{min} increases as the internal population (of 50% of the wires) decreases. The three architectures listed above require roughly the same transistor area, but the 100% connection-block population, 60% switch-block population architecture requires the lowest number of routing tracks per channel and hence the least metal area. This probably makes it slightly preferable to the other two architectures.

Figure 7.17 shows why we would intuitively expect an FPGA that contains some length 4 wires with 60% switch-block population to have good routability. These wires can always be interconnected so that two sides of any logic block in the FPGA can be reached from any point. Since the input and output pins on our logic blocks

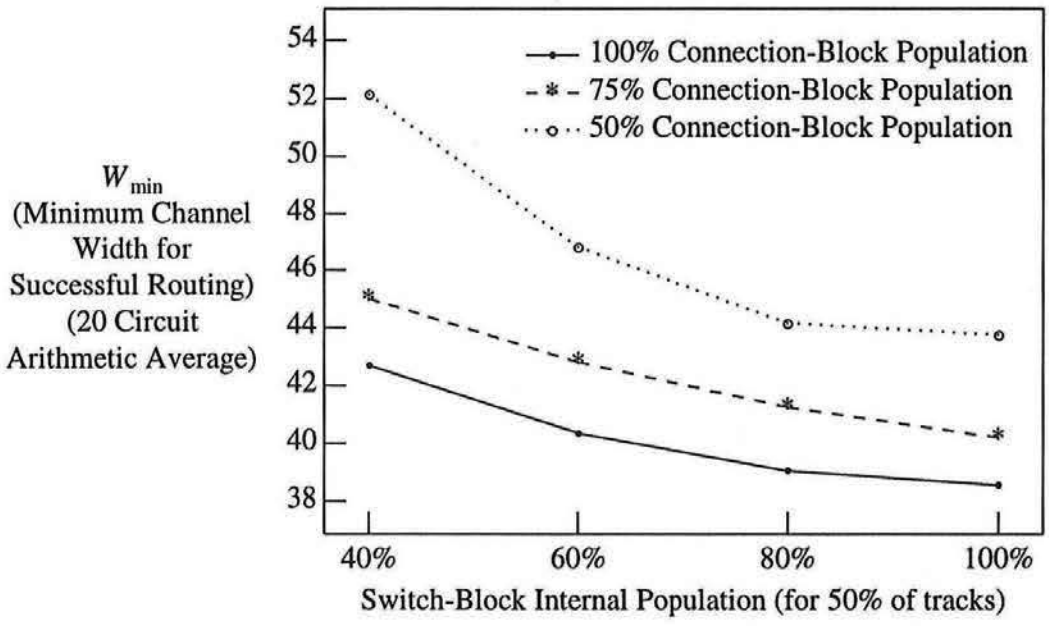


FIGURE 7.16 W_{min} for architectures with 50% of routing tracks depopulated in some manner.

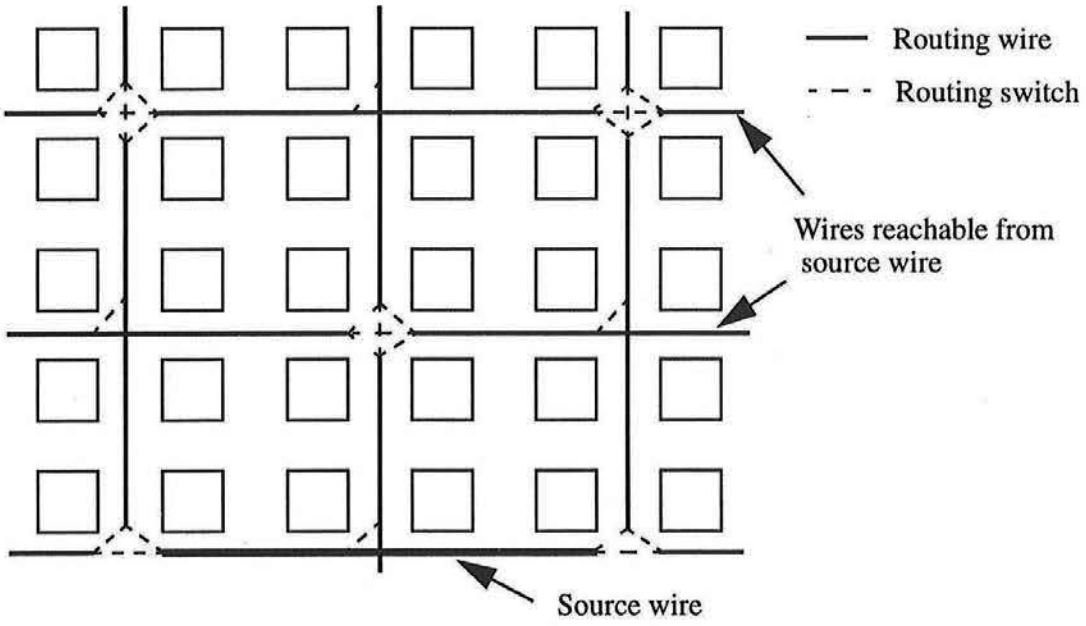


FIGURE 7.17 Length 4 wires with 60% switch-block population can reach 2 sides of any logic block.

are all logically-equivalent, a connection can use any logic block input or output pin. Consequently, as long as all the pins on the "reachable" sides of a target logic block are not already used, the connection can be made to one of these "reachable" sides. Some wiring tracks must have higher than 60% internal population, however, so that they can reach any side of a logic block and route connections when all the pins on one side of a logic block have already been used. Note that wires with lower switch-block population (e.g. 40%) cannot reach *any* side of some logic blocks from some points, significantly reducing routing flexibility.

In the experiments above, we arbitrarily decided that only 50% of the routing tracks would be internally depopulated in some way, while the other 50% of routing tracks were fully populated. We found that for such architectures, a connection-block population of 100% and a switch block population of 60% was a good choice for the depopulated wire segments. In Figures 7.18 and 7.19, we vary the proportions of these two types of wire segments: some wire segments have a switch-block population of 60% while the remaining wire segments have a switch block population of 100%. All wire segments have a connection-block population of 100%.

Figure 7.18 shows the how speed varies as the fraction of tracks that have only 60% internal switch-block population is varied. Again, the speed increase is small — only about 2%. Notice also that essentially all of this speed gain has already occurred when only 17% of the routing tracks have been internally-depopulated. This indicates

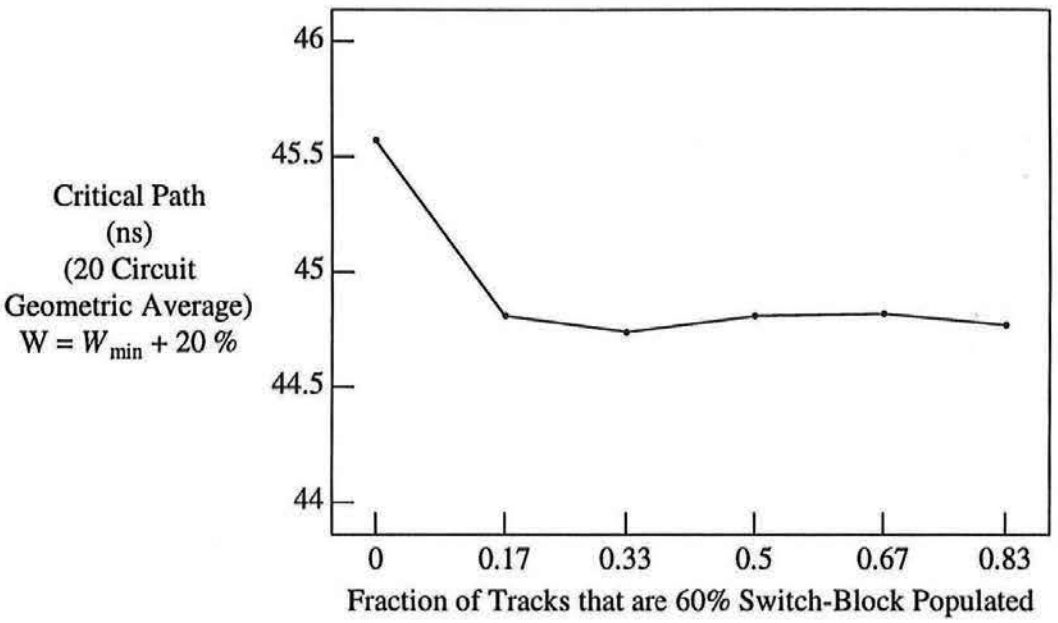


FIGURE 7.18 Speed versus the fraction of routing tracks with a switch-block population of 60%.

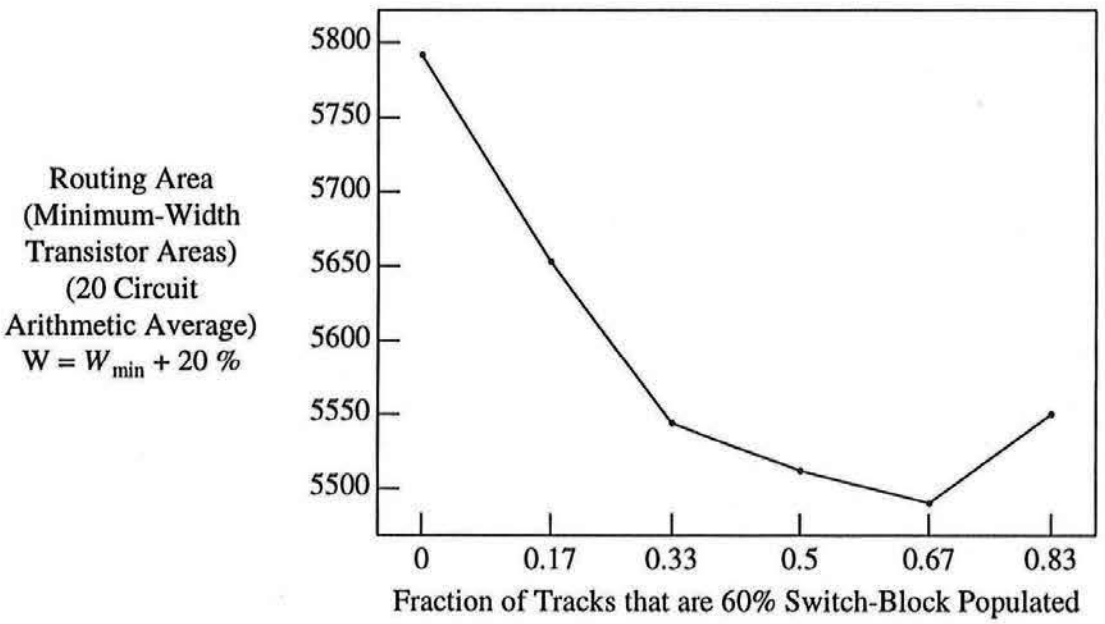


FIGURE 7.19 Area versus the fraction of routing tracks with a switch-block population of 60%.

that our router is able to correctly identify the time-critical nets and move them onto the fastest resource available.

Figure 7.19 shows how routing area varies with the fraction of tracks that have only 60% internal switch-block population. The best area occurs when 67% of the routing tracks have been depopulated in this way — any more and the increase in tracks required to successfully route a circuit outweighs the reduction in the average number of switches per wire. Using a switch-block population of 60% for two-thirds of the routing tracks yields a 5.2% reduction in routing area versus fully populating all routing tracks.

7.5.2 Two-Wire-Type Architectures

To determine if the routing architectures of Section 7.4 could also benefit from reductions in the internal population of some routing wires, we conducted experiments in which we reduced the internal population of some of the best architectures found in Section 7.4. Figure 7.20 shows how area-efficiency varies as the fraction of routing tracks using a switch-block internal population of 60% is varied, for a representative architecture. The routing architecture shown in Figure 7.20 contains 33% length 4 buffered wires and 67% length 4 pass-transistor-switched wires. The connection-block population of all wires is 100%. The three curves in Figure 7.20 correspond to

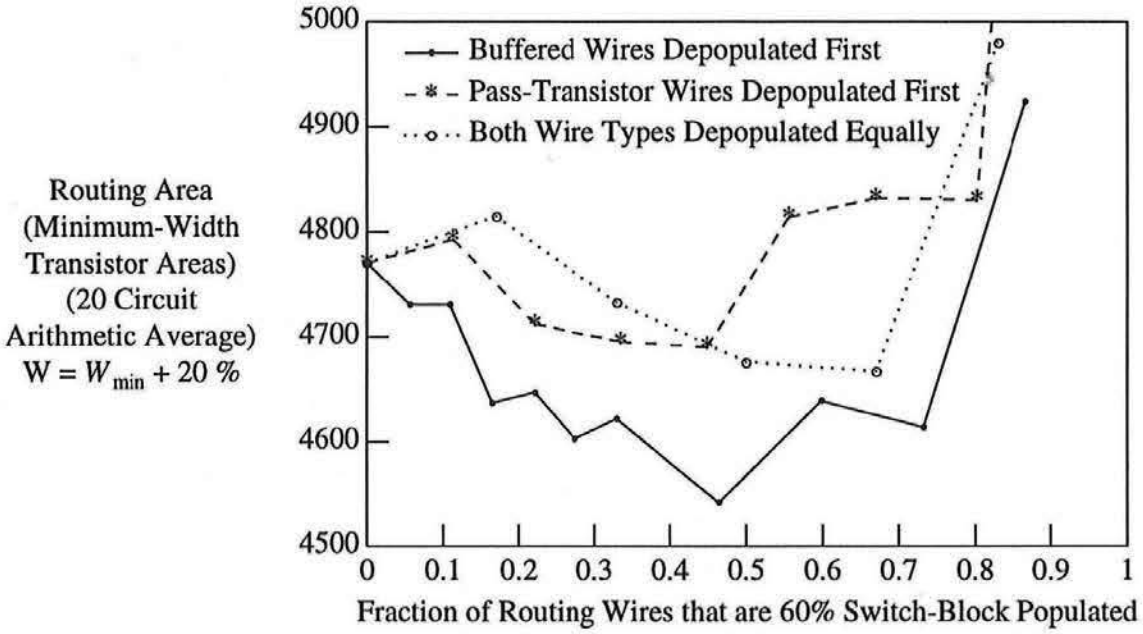


FIGURE 7.20 Area versus fraction of routing tracks with 60% switch-block population. Architecture contains 33% buffered length 4 wires and 67% pass-transistor-switched length 4 wires.

three different ways to depopulate the routing wires. The solid curve depopulates the buffered wires first, and depopulates the pass-transistor-switched tracks only once all the buffered wires are depopulated. The 0.33 point on this curve therefore corresponds to an architecture in which all the buffered routing tracks have a switch-block internal population of 60%, and all the pass-transistor switched tracks have a switch-block population of 100%. The dashed line, on the other hand, depopulates the pass-transistor-switch wires first. The 0.67 point on this curve, then, corresponds to an architecture in which all the pass-transistor-switched wires have a switch-block population of 60%, and all buffered wires have a switch-block population of 100%. Finally, the dotted line depopulates pass-transistor-switched and buffered wires in equal proportion. For example, the 0.67 point on this curve denotes an architecture in which 67% of the buffered wires and 67% of the pass-transistor-switched wires have a switch-block population of 60%; the other 33% of the routing wires have a switch-block population of 100%. Note that the "0" point on the horizontal axis corresponds to an architecture in which all routing wires are fully switch-block populated, for all three curves.

From Figure 7.20 one can clearly see that depopulating buffered routing tracks first leads to the best area-efficiency. This makes intuitive sense — since tri-state buffer routing switches consume more area than pass transistors, eliminating these buffered switches saves more area than eliminating pass transistors. The best area-efficiency occurs near the 0.5 point on the solid curve of Figure 7.20. In this architecture all the buffered wires (33% of routing tracks) have a switch block internal population of 60%. 20% of the pass-transistor-switched wires (13% of the routing tracks) also have a switch-block population of 60%, while the other pass-transistor-switched wires (54% of the routing tracks) have a switch-block population of 100%. This architecture has 4.8% better area-efficiency than an FPGA in which all routing tracks are fully switch-block populated. Depopulating routing wires for this architecture did not produce a significant speed gain, however.

We have experimented with other two-wire-type FPGA architectures, and found that they also tend to have the best area-efficiency when the buffered routing tracks have a switch block only in every second potential location. Depopulating some of the pass-transistor-switched wires as well can lead to a small further area improvement if less than 50% of the routing tracks consist of buffered wires. For example, an architecture containing 50% length 8 buffered wires and 50% length 4 pass-transistor-switched wires has 5.3% better area-efficiency when the buffered wires have a switch-block population of 56% (i.e. 5 out of 9 potential switch blocks) than when all wires are fully switch-block populated.

7.6 Wire Spacing for Speed

The previous sections considered optimizing the topology of an FPGA's routing architecture — i.e. the connectivity of the switches and wires. In this section we consider a purely electrical optimization: varying the spacing between routing wires. In modern IC processes, the coupling capacitance between adjacent wires in the same metal layer is a large fraction of the total wire capacitance. Increasing the spacing between wires reduces this coupling capacitance, and hence speeds up an FPGA's routing. For example, in TSMC's 0.35 μm process, increasing the spacing between metal 3 wires from the minimum value (0.5 μm) to five times the minimum value (2.5 μm) reduces the coupling capacitance between adjacent wires to almost nothing, and reduces the total metal capacitance by over 48%. Of course, this increase in metal spacing reduces metal routing density — the pitch of the wires is increased by a factor of 2.8. Often, therefore, it will not be possible to increase the metal spacing of every routing wire this much without requiring more metal area than is available in the FPGA layout (i.e. all the required metal would no longer fit above the area required by the transistors, so the chip layout would have to be expanded).

A natural question then, is whether it is necessary to reduce the capacitance of every routing wire in this way to speed up an FPGA, or if reducing the capacitance of only a subset of the routing wires is sufficient. Figure 7.21 shows how the speed of a length 4 buffered architecture (the best single-wire-length FPGA found in Section 7.3.2) varies as the fraction of routing tracks that are widely spaced is varied. In Figure 7.21 all

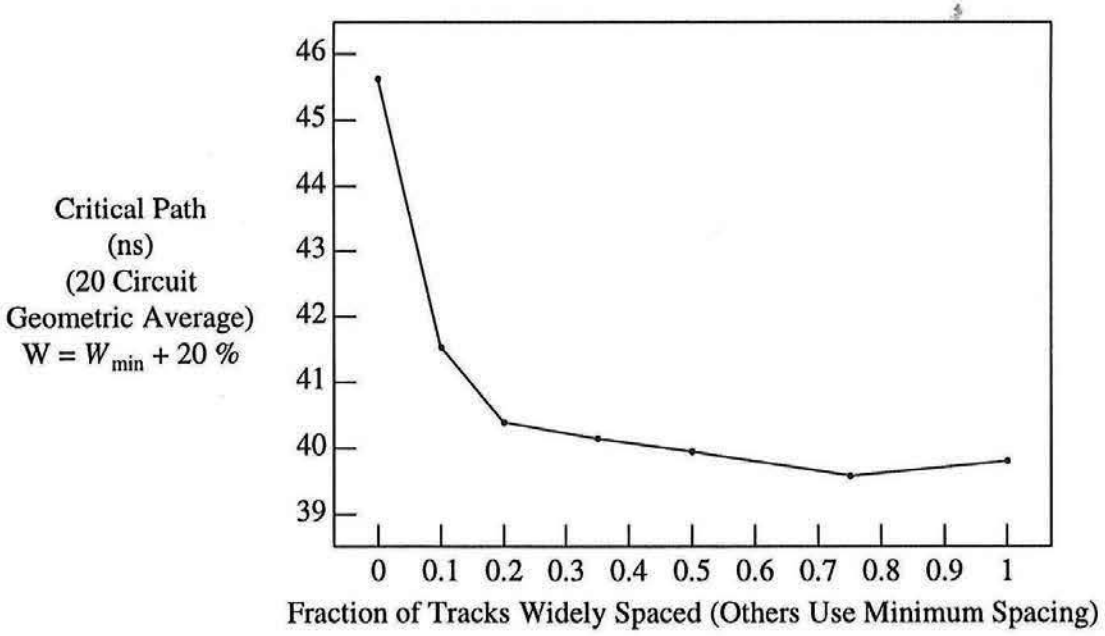


FIGURE 7.21 Speed of a length 4, buffered wire FPGA vs. routing wire spacing.

routing wires have length 4, and connect via tri-state buffers. The "0" point on the horizontal axis corresponds to an FPGA that uses the minimum metal spacing between all the routing tracks, while the "1" point corresponds to an FPGA that using a wide (2.5 μm) spacing between all the routing tracks. Observe that increasing the metal spacing significantly improves speed — an FPGA that widely spaces all the routing wires is 14.6% faster than an FPGA that uses the minimum spacing for all the wires. Even more interesting is the fact that we can obtain most of this speedup by increasing the spacing between only 10% to 20% of the routing wire segments. If we use a wide spacing between only 10% of the routing wires, we obtain a 9.8% speedup, and widely spacing 20% of the routing wires yields a 12.9% speedup. By increasing the spacing between only 10% to 20% of the routing wires we minimize the increase in metal area required, but still obtain almost all the achievable speedup.

Note that Figure 7.21 also provides an indirect validation of our router. If our router used the faster wires equally for all connections, Figure 7.21 would be a straight line from the slowest architecture (when all routing wires use the minimum spacing) to the fastest architecture (when all routing wires are widely spaced). Instead, however, we obtain most of the achievable speedup when only 10% to 20% of the wires are widely spaced. This implies that our router is able to correctly identify the most time-critical nets and route them via the fastest routing resources.

Figure 7.22 shows how the speed of one of the best two-wire-type architectures varies with the spacing between routing wires. In this architecture 50% of the routing wires

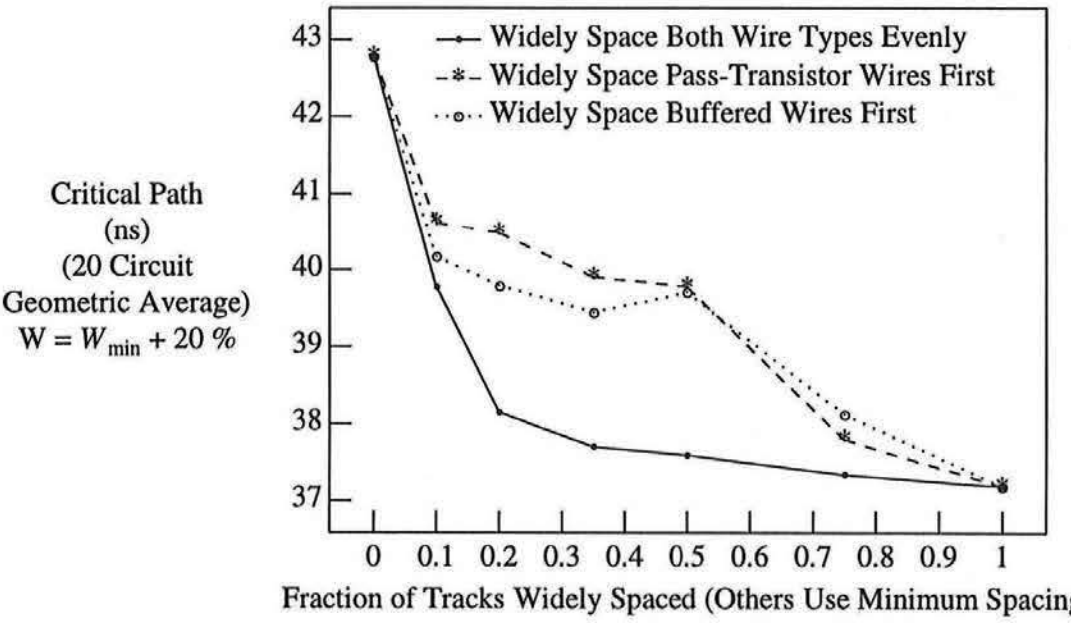


FIGURE 7.22 Speed of a length 4, 50% buffered, 50% pass-transistor FPGA vs. wire spacing.

are length 4 wires connected by tri-state buffers, while the other 50% are length 4 wires connected by pass transistors. Again, the horizontal axis denotes the fraction of tracks that are widely spaced (2.5 μm spacing), while the other tracks use the minimum metal spacing. The solid line shows the speed obtained when the widely spaced tracks are allocated evenly to buffered and pass-transistor-switched wires. The dashed line is obtained when we increase the spacing of the pass-transistor-switched tracks first, and increase spacing of the buffered tracks only after all the pass-transistor-switched tracks have wide spacing. The dotted line occurs when we do the opposite — we first increase the spacing of the buffered tracks, and increase the spacing of the pass-transistor-switched wires only after all the buffered wires are widely spaced.

Once again, increasing the spacing of all the metal tracks leads to a significant speedup; in this case the speedup is 15.1%. The solid line in Figure 7.22 shows that most of this speedup is again obtained when only 10% to 20% of the routing tracks are widely spaced. In this case, we obtain a speedup of 7.6% by spacing out only 10% of the routing tracks, and a speedup of 12.1% by spacing out 20% of the routing tracks. Comparing the solid line in Figure 7.22 to the other two lines shows that one must increase the spacing of both some pass-transistor-switched wires and some buffered wires. About half the total speedup is due to speeding up the pass-transistor-switched wires, and the other half is due to speeding up the buffered wires. This indicates that both these types of wire are useful for making speed-critical connections — the pass-transistor-switched wires are better for short connections, while the buffered wires are better for long connections.

While this section has examined speeding up an FPGA by spacing out some of its wires, one can achieve similar effects by increasing the size (and hence the drive strength) of the routing switches used to interconnect wires on certain routing tracks. From the results of this section, we can predict that increasing the size of the switches on only 20% of the routing tracks will produce almost as much speed gain as increasing the switch size on every routing track.

7.7 Overall Architecture Comparison

In the previous sections we have examined FPGA routing architectures of gradually increasing complexity and looked for important architectural trends by varying the key architectural parameters. In this section we provide an overview of all the results of this chapter by comparing some of the best architectures found in each of the prior sections against each other and against a routing architecture that is similar to that of the popular Xilinx XC4000X series FPGAs [139, 140]. As described in Section 4.6.2, this "4000X-like" architecture contains 25% length 1 wires, 12.5% length 2

wires, 37.5% length 4 wires, and 25% "one-quarter longs", whose length is one-fourth of the chip. The length 1 and 2 wires connect via pass transistors, while the longer wires connect via tri-state buffers. As well, pass transistor switches also allow the length 4 wires to connect to the length 1 and 2 wires, and the one-quarter longs to connect to length 1 wires. While this routing architecture is very similar to that of the Xilinx XC4000X, it simplifies a few features [140].

Table 7.7 compares the speed and density of some of the best architectures found in each of the preceding sections to those of this 4000X-like architecture. We also include the performance of an FPGA composed entirely of length 1 wires connected by pass transistors, since most prior FPGA research has focused on this architecture. All the architectures below have a connection-block internal population of 100%, and all routing wires are minimum width, minimum spacing metal 3 wires. All the architectures have an $F_{c,ouput}$ of 0.25·W and an $F_{c,input}$ of 0.5·W.

TABLE 7.7 Comparison of key architectures (20 circuit average).

Segmentation of Routing Tracks, and Switch Types Used	Switch-Block Internal Population	Delay (ns)	Speedup vs. 4000X-like FPGA	Routing Area (Min. Width Transistor Areas)	Routing Area vs. 4000X-like FPGA
Xilinx 4000X-like: 25% L1, 12.5% L2, 37.5% L4, 25% one-quarter longs; mix of buffers and pass transistor switches	100% for all wire lengths except L2. For L2, 66%.	48.83	—	4425	—
100% L1, pass-transistor switched	Not relevant	120.7	-60%	5891	+33.1%
100% L8, pass-transistor switched	100%	46.22	+5.6%	5131	+16.0%
100% L4, buffer switched	100%	45.57	+7.2%	5792	+30.9%
100% L4, buffer switched	60% for 67% of tracks; 100% for 33% of tracks	44.82	+8.9%	5490	+24.1%
67% L4, pass-transistor switched; 33% L4, buffer switched	100%	42.91	+13.8%	4771	+7.8%

TABLE 7.7 Comparison of key architectures (20 circuit average).

Segmentation of Routing Tracks, and Switch Types Used	Switch-Block Internal Population	Delay (ns)	Speedup vs. 4000X-like FPGA	Routing Area (Min. Width Transistor Areas)	Routing Area vs. 4000X-like FPGA
83% L4, pass-transistor switched; 17% L4, buffer switched	100%	44.31	+10.2%	4569	+3.3%
50% L4, pass-transistor switched; 50% L8, buffer switched	100%	41.04	+19.0%	5039	+13.9%
83% L4, pass-transistor switched; 17% L8, buffer switched	100%	43.84	+11.4%	4539	+2.6%
67% L4, pass-transistor switched; 33% L4, buffer switched	60% for buffer-switched wires; 100% for pass-transistor-switched wires	43.15	+13.2%	4622	+4.5%
50% L4, pass-transistor switched; 50% L8, buffer switched	56% for buffer-switched wires; 100% for pass-transistor-switched wires	41.23	+18.4%	4708	+6.4%
83% L4, pass-transistor switched; 17% L8, buffer switched	56% for buffer-switched wires; 100% for pass-transistor-switched wires	44.04	+10.9%	4426	0%

The architectures are listed (after the 4000X-like architecture) in order of increasing complexity. Notice the extremely poor performance of an FPGA using only length 1 wires connected via pass transistors — 147% slower (-60% speedup) and 33.1% percent larger than the 4000X-like architecture. The best architecture we found using only pass transistors and one length of wire used length 8 wires. This architecture performs much better than a length 1 architecture; it is 5.6% faster than the 4000X-like architecture, at a cost of 16% larger area than that of the 4000X-like architecture. Although the speed and area-efficiency of this length 8, all pass-transistor FPGA are reasonably competitive on average, we consider FPGA architectures that contain no

buffers dangerous. As circuit size increases, the longest connections in an FPGA grow longer, and will pass through more series switches. Since the delay of pass-transistor switches grows quadratically with the number of switches in series, it is difficult for an architecture that contains only pass transistors to maintain good speed as the size of the logic block array grows. As well, larger circuits can contain nets with a higher maximum fanout, and purely pass-transistor based routing is inefficient for routing high-fanout nets. For both these reasons, architectures that contain only pass transistors do not scale as well with increasing circuit size as architectures that contain some buffers.

The best single-wire-type architecture we found, in which all wires are length 4 and all switches are buffers, is 7.2% *faster* than the Xilinx 4000X-like FPGA. Its area is 30.9% larger, however. It is interesting that such a simple FPGA architecture is reasonably competitive with the complex routing architecture of the 4000X-like FPGA. Simpler routing architectures make it easier to develop CAD tools. As well, they likely make it easier to implement intellectual-property "cores." These cores are sometimes provided as "hard" (placed-and-routed) macros. If there is only one type of routing resource in the FPGA, it is easier to map several of these hard cores into one FPGA and ensure each gets the wires it needs. These factors may make a simple FPGA architecture, in which all the wires are essentially the same, attractive despite its suboptimal speed and area performance.

By reducing the internal switch-block population of some of the wires in this all length 4 architecture to 60% we can reduce the area overhead versus the 4000X-like architecture from 30.9% to 24.1%.

Table 7.7 also lists four of the best two-wire-type architectures. Each of these architectures combines some buffered wires with some pass-transistor-switched wires. Two of these architectures use only length 4 wires, while the other two use some length 4 and some length 8 wires. Notice that these architectures are all significantly (10.2% to 19%) faster than the 4000X-like architecture, but none of them is as area-efficient. The area penalty for using an architecture that is 19% faster than the 4000X-like architecture is 13.9%, while the area penalty for an architecture that is 11.4% faster than the 4000X-like architecture is only 2.6%. Both the speed and the area of these architectures are significantly better than the best single-wire-type architecture discussed above, showing that a mix of pass transistors and buffers is very useful in FPGA routing.

The last three lines in Table 7.7 show the benefits of reducing the switch-block population of these two-wire-type architectures so that the buffered wires have a switch block only once every two logic blocks. These architectures are from 2.5% to 7.5% more area-efficient than those that used a switch-block population of 100% for all

wires; the exact amount of area improvement depends on what fraction of the routing wires use buffered switches. One of these architectures is 18.4% faster than the 4000X-like architecture and only 6.4% larger, and another architecture is 10.9% faster and uses the same area as the 4000X-like architecture.

From the results of Table 7.7 one can see that we have found many architectures with speed superior to that of the 4000X-like architecture, but none with superior density. We believe the 4000X contains too many short wire segments, and this reduces its speed versus many of the architectures we have investigated. The 4000X uses a more advanced switch block than the disjoint switch block used by all the other architectures in Table 7.7, however. The 4000X switch block contains some switches that allow wires of different lengths to connect, and that allow wires in different tracks to connect. These features make the 4000X switch block more routable than the disjoint switch block, yet it contains only a few more switches than the disjoint switch block. Consequently, this switch block tends to result in FPGAs with superior density. As discussed in Section 7.3.1, we consider the investigation of better switch block topologies for use with FPGAs that contain some long wires to be a fertile area for future research. We expect that combining the segmentation distributions of some of the architectures listed in Table 7.7 with a better switch block would lead to significantly improved area-efficiency.

Finally, note that the speed enhancement of increasing the spacing between 20% or so of the routing wires (described in Section 7.6) can be applied to any of the architectures listed above to yield a further speed increase of approximately 13%.

7.8 Summary

In this chapter we have investigated a large number of different detailed routing architecture issues. First, we showed that a disjoint switch block topology is superior to a Wilton switch block topology for FPGAs that contain wires longer than one logic block. The switch block used in the Xilinx XC4000X FPGA is superior to either of these alternatives, however, and considerable further research is needed in this area.

Secondly, we showed that it is most important for FPGAs to contain wires of moderate length (4 to 8 logic blocks). While most commercial FPGAs contain some very short and some very long wires, we have found FPGAs that use significant numbers of these types of wires to be inferior to those that employ medium-length wires.

We also found that FPGAs that contain a mix of pass-transistor and tri-state buffer routing switches are superior to FPGAs that employ only one type of switch. The

fastest FPGAs tend to contain about 50% pass-transistor switches and 50% tri-state buffer switches, while the most area-efficient FPGAs contain about 80% pass-transistor switches and only 20% tri-state buffer switches.

We also found that reducing the switch-block internal population of routing wires that interconnect with tri-state buffers produces an area gain of 2.5% to 7.5% for typical architectures. The switch block population of these buffered wires should be set so that each wire contains a switch block at only every second potential location along its length. Reducing the connection-block internal population of long wire segments did not produce any significant area gain, however.

We showed that by increasing the metal spacing of only about 20% of the routing wires, one can increase the speed of an FPGA by over 12%. Increasing the spacing between wires reduces capacitance, and our router can route almost all the time-critical nets on only 20% of the routing tracks. Consequently, increasing the spacing of 20% of the routing wires yields almost as large a speedup as increasing the metal spacing on all the routing wires.

Finally, we showed that the best architectures examined in this chapter have significantly superior speed to a (slightly simplified) Xilinx XC4000X routing architecture; some architectures are 19% faster than the 4000X architecture. This 4000X-like routing architecture has better area-efficiency than all but one of the architectures we examined, however. The best architectures in this chapter have a better area-delay product than the 4000X-like architecture, indicating that they have gained more in speed than they have sacrificed in area. They are also less complex than the 4000X. This is a useful feature, as it simplifies both CAD tool design and the implementation of pre-placed and pre-routed intellectual property cores.

CHAPTER 8 *Conclusions and Future Work*

8.1 Summary and Contributions

This work has contributed to two related research areas: FPGA CAD algorithms and FPGA architecture. The new CAD algorithms and tools developed in this research were described in Chapters 3 and 4, and are briefly summarized in Table 8.1. In Chapter 3, we developed the first publicly-described logic block packing tools targeting cluster-based logic blocks.[1] We also created a new simulated annealing based placement tool (the placement portion of our Versatile Place and Route (VPR) program) which incorporates three new enhancements over prior tools. First, we implemented a new annealing schedule that adapts automatically to different placement problems, provides good result quality and is more robust than the schedule of Huang et al [82]. Second, we developed a new, "linear congestion" placement cost function that enhances the routability of circuits mapped to FPGAs in which different channels have different widths. Finally, we developed an incremental net bounding box update algorithm that reduces the CPU time required for placement by more than a factor of five, on average, vs. using the traditional brute-force bounding box recomputation.

Chapter 4 described new routing algorithms and a parameterized FPGA routing architecture generator that were implemented in the VPR placement and routing program. We described a new problem in FPGA CAD: the automatic generation of *good* rout-

1. Altera Corporation has an internal logic block packing tool that targets logic clusters, and Xilinx has a tool targeting logic blocks very similar to logic clusters for use with the XC5200 FPGA. The algorithms used by these tools have never been disclosed, however.

TABLE 8.1 Summary of CAD contributions.

CAD Area	Contributions
Logic Block Packing	• First academic tools to target parameterized cluster-based logic blocks
Placement	• Architecture independent placement tool • New, robust annealing schedule • Fast incremental update scheme • High quality results
Routing	• Architecture-independent router • Architecture generator for ease of specification • Fully timing-driven • Directly optimizes the Elmore delay • Very high quality
Modelling	• Transistor-based area model • Full delay extraction • Path-based timing analysis

ing architectures from a succinct list of understandable parameters. We showed that it is often difficult to simultaneously satisfy all the specified parameters, and highlighted some of the novel features of our architecture generator that allow all the architecture parameters to be satisfied under most circumstances, and result in an FPGA with good routability.

Chapter 4 also described the two routers we developed and implemented in VPR: one is purely routability-driven, while the other is both routability- and timing-driven. The routability-driven router is based on the Pathfinder negotiated congestion algorithm [85] but includes enhancements to the routing cost function and "routing schedule." As well, the VPR routability-driven router employs a new method we developed for the breadth-first routing of multi-terminal nets which reduces the CPU time to route high-fanout nets by an order of magnitude compared to the traditional breadth-first maze router method. In terms of circuit routability, this routability-driven router outperforms all other routers to which we can compare, and the combination of the VPR placement algorithm and the VPR routability-driven router significantly outperforms all combinations of placement and routing tools to which we can compare.

The VPR timing-driven router employs new algorithms we developed to directly optimize the Elmore delay, while simultaneously resolving routing congestion. Previous non-commercial FPGA routers have optimized either wirelength or the (inaccurate) linear delay model; by directly optimizing the more complex Elmore delay model our router can make much more intelligent routing decisions. As well, our timing-driven

router is the first non-commercial "buffer-aware" FPGA router,[1] and it performs buffer insertion automatically as it constructs net routing topologies. This timing-driven router also has excellent performance: it is capable of optimizing for routability almost as well as our routability-driven router, while producing circuits which are 2.6 times faster, on average. The run-time of this router is also quite low — it can route an 8300 LUT circuit in 3.2 minutes.

As well as the new algorithms incorporated in VPack, T-VPack and VPR, we consider the VPR / (T-)VPack CAD infrastructure itself to be a major contribution of this work. VPR not only produces high-quality placements and routings, it is also very flexible; i.e. it can target a wide variety of FPGA architectures. VPR includes fast net delay extraction and path-based timing analysis modules, allowing easy evaluation of the speeds of different FPGA architectures. It also includes a detailed, transistor-based area model, allowing easy comparison of the area of different FPGA architectures. Taken together, these capabilities enable more sophisticated and detailed comparison of FPGA architectures than were possible with previous tool sets. In addition to the three architectural studies described in this book, VPR has been used to evaluate the quality of synthetic benchmark circuits [147, 148, 149], to investigate laser-programmed gate array architectures [150], to generate good global routings for use with an FPGA detailed router [101], and as a CAD infrastructure for research into ultra-fast FPGA routing algorithms [151, 127, 152] and ultra-fast FPGA placement algorithms [153]. Ongoing research at the University of Toronto is enhancing VPR in order to research the routing architecture of Altera-like (hierarchical) FPGAs [138]. Several FPGA companies are currently using VPR to aid in the development and evaluation of new FPGA architectures, and researchers from over 80 universities and 30 companies have downloaded copies of VPR from our web site. VPR, VPack and T-VPack are all available for download from *http://www.eecg.toronto.edu/~jayar/software/software.html*, and may be freely used for academic FPGA research.

In chapters 5, 6 and 7 we used these VPR, VPack, and T-VPack CAD tools to investigate three different FPGA architecture issues; Table 8.2 briefly summarizes our most important architecture conclusions. In Chapter 5 we explored FPGA global routing architectures, which specify the relative widths of the various channels within an FPGA. We found that the best global routing architecture distributes the logic block pins evenly around the logic block perimeter, and has the same width for every channel within the FPGA. Row-based FPGAs, which have logic block pins only on the top and bottom of each logic block, are only 8% less area-efficient than this best architecture, however, as long as the horizontal channels are twice as wide as the vertical channels. We also found that by adjusting the relative widths of the horizontal

1. The routing tools of Xilinx Inc. are buffer-aware [107], and those of other companies whose FPGAs contain a mix of pass transistors and buffers are also likely buffer-aware.

and vertical channels, one can create FPGAs with rectangular logic block arrays which are nearly as area-efficient as FPGAs with square logic block arrays. We showed that making the routing channel between the I/O pads and the logic block array moderately (25%) wider than the other channels is a net benefit when the I/O positions are locked. We showed that making one or more routing channels near the center of the array extra-wide, as some commercial FPGAs do [141], is not very useful, however. Although these issues have been widely discussed and speculated upon, the work presented here is the first systematic study of global routing architecture.

TABLE 8.2 Summary of FPGA architecture conclusions.

Architectural Issue	Key results
Global routing architecture	• Uniform channels better than non-uniform channels • No directional bias is best
Cluster-based logic blocks	• Best cluster size: 4 - 10 look-up tables • Better speed, area and compile time than single-BLE logic block • Need only about half as many cluster inputs as there are LUT inputs • Should reduce logic to routing interconnect flexibility for large clusters
Detailed routing architecture	• Relatively simple architectures perform well • Length 1 wires not useful • Best single wire segment length = 4 • 50% to 80% of routing switches should be pass transistors; remainder tri-state buffers • Slight switch-block depopulation yields area gain • Decreasing the delay of only 20% of the routing wires yields the same circuit speed as decreasing the delay of 100% of the routing wires

In Chapter 6, we presented an extensive study of "cluster-based" logic blocks, which are logic blocks that contain several LUTs and registers interconnected by local routing. While Altera FPGAs use cluster-based logic blocks, this is the first study of how one should choose the key logic cluster parameters, and of the speed and area-efficiency achievable with these logic blocks. We investigated several key logic cluster issues. First, we found that one need not provide $4N$ distinct inputs to a cluster of N 4-LUTs — approximately $2N+2$ distinct inputs suffice to achieve full logic utilization, and result in a more area-efficient logic block. Second, we found that as the size of the logic cluster increased, one should reduce F_c to achieve the best area-efficiency; a good $F_{c,output}$ value is W / N, while the best F_c value for logic block inputs is somewhat higher. Third, we found that logic clusters containing between 4 and 10 LUTs have the best area-delay product. An FPGA employing a logic cluster of size 7, for

example, requires 19% less area, achieves 21% higher circuit speed, and has a 33% lower area-delay product than an FPGA with a single-BLE logic block. Finally, we found that the use of larger logic clusters significantly reduced the time required to place and route designs. For example, use of a logic cluster of size 10 reduced the time to place and route circuits by a factor of 5.5 vs. using a single-BLE logic block.

In Chapter 7, we investigated FPGA detailed routing architecture, which specifies the exact pattern of wires and switches forming the routing fabric, for island-style FPGAs. While some detailed routing architecture questions have been investigated before, this study is the first to use a truly timing-driven router, the first to evaluate an architecture's speed via the critical path delay, and the first to model routing area using a new minimum-width transistor equivalent area model. As well, many of the detailed routing architecture issues we investigated had not been previously studied. More specifically, our work is the first to consider routing that contains tri-state buffer routing switches, partially-internally depopulated routing wires, and routing wires with varying metal spacing.

One of the most important conclusions of this study is that most of the wires in an FPGA should be of moderate length (4 to 8 logic blocks); the very short and very long wires often found in commercial island-style FPGAs lead to inferior speed and area-efficiency. As well, we found that FPGAs should contain a mix of tri-state buffer and pass transistor routing switches; using pass transistor switches on approximately 50% to 80% of the routing tracks is best. Internally depopulating wire segments that use tri-state buffer routing switches so that they have a switch block in only every second location leads to a 2.5% to 7.5% area-efficiency gain. Neither reducing the switch-block population of wires that use pass transistor switches nor reducing the connection-block population of wires is as successful at reducing routing area. We also found that increasing the spacing between metal wires significantly improved the FPGA speed, and that one could obtain virtually the entire speedup by increasing the spacing of only 20% of the routing wires. Specifically, increasing the spacing of 20% of the routing wires typically improves FPGA speed by 12% to 13%, while increasing the spacing of all the wires increases the speed by about 15%. Finally, we found that properly designed routing architectures employing only two different lengths of wire and a simple switch block lead to 11 - 18% better speed than a routing architecture similar to that of the Xilinx XC4000X FPGA, with only a 0 - 6% area-efficiency penalty. These architectures are considerably simpler than that of the Xilinx XC4000X, and this makes them even more attractive, since simple architectures ease the task of writing CAD tools and implementing "hard" intellectual property cores.

8.2 Future Work

8.2.1 CAD Tool Enhancements

There are two different ways in which one can enhance our CAD tools: by improving the core algorithms to increase result quality, and by increasing the flexibility of the FPGA architecture generator to allow easy investigation of a wider class of FPGAs.

In this work we have created timing-driven logic block packing and timing-driven routing tools, but the placement phase of VPR is purely routability-driven. Accordingly, the VPR placement algorithm should be enhanced to make it timing-driven — work is already underway on this project at the University of Toronto [19]. Another interesting project would explore whether incorporating more advanced delay models into T-VPack (such as fanout-based delay estimates) would improve circuit speed by giving T-VPack a better estimate of where the post-place-and-route critical path will lie.

The VPR placement algorithm could also be enhanced by adding a cost function that considers the underlying detailed routing architecture when evaluating the cost of a placement. For example, in an FPGA that contains only length 4 wires, any connection between logic blocks in the same row or column is likely to be completed with one wire segment if it spans less than four logic blocks. If, however, the connection spans five logic blocks, or if the connection is between logic blocks that are not in the same row or column, it must use more than one wire segment. Consequently, the cost function should consider not only wirelength, but also the number of wire segments (and for timing-driven placement, the minimum delay) required to route from a net source to a net sink. Using a cost function of this type results in a "routing-aware" placement algorithm, without the high CPU time required by simultaneous placement and routing approaches.

The other major class of CAD enhancements concern creating a wider variety of FPGA architectures automatically. As discussed in Chapter 4, the architecture generator in VPR can build routing-resource graphs for detailed routing only when the segmentation distribution in the horizontal and vertical channels is the same. While this is generally true of island-style FPGAs, it is not true of the hierarchical style of FPGAs used by Altera. A research project is currently under way at the University of Toronto to remove this restriction, and hence allow the generation and evaluation of Altera-style FPGAs [138].

Automatically generating good FPGA architectures to match a set of parameters is a new problem in FPGA CAD, and there are many avenues for future research in this

area. While we have found ways to simultaneously satisfy all the specified parameters in many cases, in some situations we cannot meet all the user's specifications. An important question, then, is whether it is impossible to satisfy all the constraints in these cases, or if there is some better architecture generation algorithm which enables all the constraints to be met. If it is impossible to satisfy all the specified parameters in some cases, is there some parameterization (different than the one we have chosen) of equivalent flexibility which guarantees that any set of parameters can always be simultaneously satisfied? Another problem in automatic architecture generation concerns the automatic generation of good connection block switch patterns. While the patterns currently generated by VPR work well for a wide variety of architectures, they are not necessarily the best possible patterns.

8.2.2 Future FPGA Architecture Research

In Chapter 6 we examined only fully-connected logic clusters, but it is not clear that fully-connected local routing is best. The complexity of fully-connected local routing grows quadratically with the cluster size, so the use of a less flexible local routing structure might make larger logic clusters more attractive. On a related note, is it necessary to allow all N BLE outputs to connect to the main FPGA routing? In large logic clusters many signals are used only within the logic cluster, so the average number of outputs used is less than N. By forcing some BLE outputs to be used only with a logic cluster, an FPGA would contain fewer logic block output pin buffers and connection blocks, saving area. On the other hand, logic utilization might suffer, so the overall effect on area is unclear.

Many interesting questions about detailed routing architecture remain. Further research is needed to determine good switch block topologies for use with FPGAs that contain wires longer than one logic block. This switch block research should pay careful attention to the interaction between the distribution of the wire lengths and the performance of different switch blocks, as our results have shown these two issues to be coupled. In Chapter 7, we found that reducing the connection-block internal population of wires did not provide a significant area-efficiency gain, and that depopulating the connection block on all the wires led to unroutable FPGAs. We always depopulated the switches from logic block output pins and the switches to logic block input pins equally, however. A future research project could look into depopulating the output pin switches and the input pin switches by different amounts. Since the switches from logic block output pins to routing tracks require more area than the switches from routing tracks to input pins, depopulating the output pin switches would yield the largest reduction in switch area per wire. By depopulating the input pin switches by a smaller amount, or not at all, one could maintain good routability, and hopefully achieve a net area gain.

The best architectures we found in Chapter 7 had critical path delays of approximately 40 ns. About 80% of this delay is still inter-cluster routing delay, so there is still considerable room for speed improvement by optimizing the routing architecture. However, almost 40% of the routing delay (about 30% of the total delay) is due to the delay to get out of a logic block via an output pin buffer, and back into another logic block via a track buffer and connection-block multiplexer. Even if we could somehow create routing wires and routing switches with zero delay, then, we would only double the FPGA speed if these fixed delays to enter and leave a logic block were left unchanged. We believe research is therefore needed into "fast paths" which bypass most of the connection block multiplexer, track buffer and perhaps the output pin buffer delays; such fast connections are employed in many commercial FPGAs [4, 7]. For example, one can build direct connections (as Xilinx does) between adjacent logic blocks. These direct connections are short enough that they should not require track buffers. Fast path connections can also connect into a special, fast input in the connection block multiplexers. If the multiplexer is a tree of pass transistors (see Appendix B.1), this fast input is created by making the tree unbalanced; the fastest input has the fewest pass transistors between it and the multiplexer output. Obviously the number of fast paths one can create is very limited (if one makes too many fast paths they all become slow paths again!). Research is needed to decide if such fast paths can be used effectively enough by CAD tools to significantly speed up an FPGA, and if so, which logic blocks should connect via fast paths.

Finally, since power dissipation is of increasing concern in all VLSI design, future research should investigate detailed routing architectures and logic block architectures that lead to the best combination of not only speed and area, but also power.

APPENDIX A *Graphic Visualization in VPR*

VPR contains built-in graphics that we have found very valuable in both CAD algorithm and FPGA architecture research [134]. The graphical visualization of a new FPGA routing architecture or a routed circuit allows much quicker understanding of how well or how poorly the various algorithms in VPR are performing than trying to comprehend megabytes of textual data. On-screen buttons allow the user to pan and zoom various views of the FPGA's routing architecture and placed and/or routed circuits. PostScript output of any display on screen is always available simply by clicking on a "PostScript" button; all the figures in this appendix were generated directly by VPR in this manner. While printed here in black and white, VPR's graphics are actually in colour — see *http://www.eecg.toronto.edu/~jayar/software/software.html* for colour versions of these pictures.

Figures A.1 to A.4 show various steps in the placement and routing of the MCNC benchmark circuit e64. This circuit contains 274 four-input LUTs, and is mapped to an FPGA where every logic block is a single BLE. The FPGA routing contains a mixture of length 1, 2 and 4 wires. Figure A.1 shows a "rat's nest" view of the connected blocks before placement. The user has selected a logic block by clicking on it with the mouse; its fan-out is highlighted in dark grey, while its fan-in is highlighted in black. Notice that the fan-in and fan-out of the selected block are widely scattered across the FPGA.

Figure A.2 shows the same circuit and the same selected block after placement. Now the fan-in and fan-out of this block have been clustered quite closely around it. Most connected blocks have been placed close together, resulting in a much less cluttered view than in Figure A.1.

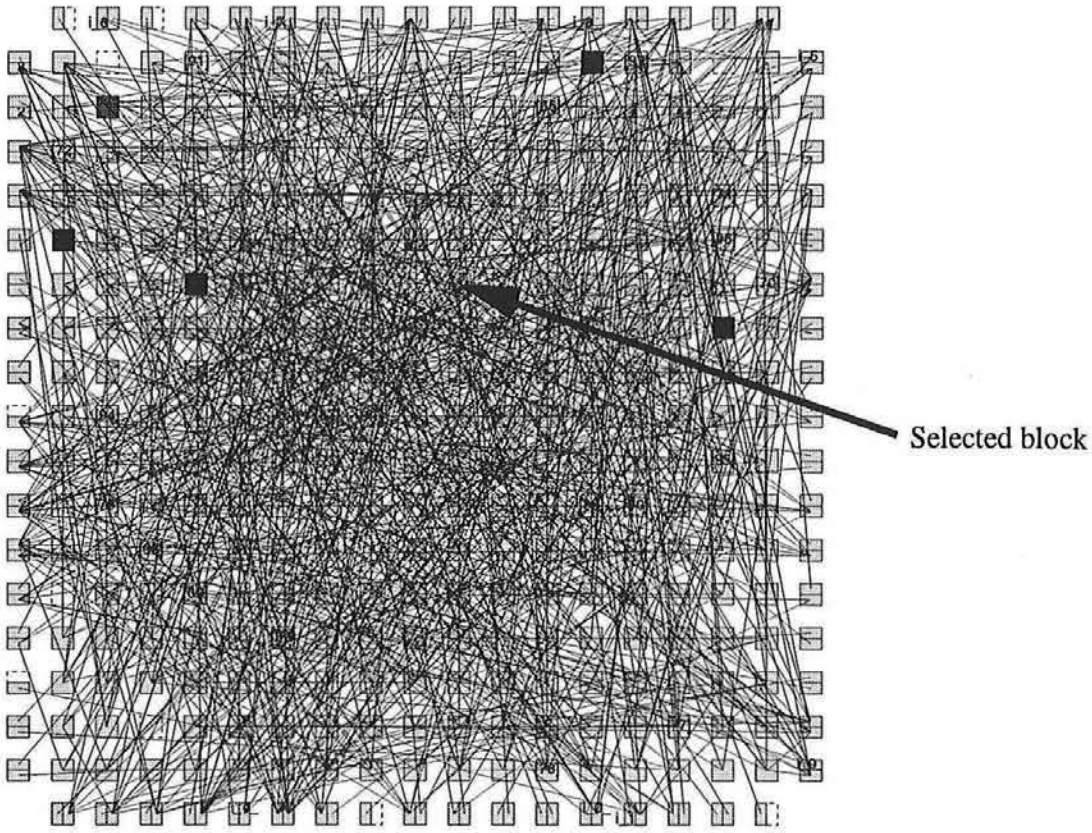


FIGURE A.1 Initial (random) placement of benchmark circuit e64.

Figure A.3 shows e64 after it has been successfully routed. By examining graphical views of routed circuits one can quickly determine how well VPR routes different classes of nets, and identify areas where the routing algorithm is not performing well.

Figure A.4 shows the circuit critical path. A user can highlight each net on the circuit critical path, one by one. Figure A.4(a) shows only the first net on the critical path — the critical connection begins at an input pad (the lighter grey block) and connects to the dark grey logic block. Figure A.4(b) shows the entire critical path, which in this case spans five nets and four logic blocks. The first four nets on the critical path are shown in dark grey. The last net on the critical path is shown in light grey; it begins at the lighter grey logic block and ends on the dark grey output pad. Being able to view the critical path of a circuit in this way has been very valuable in optimizing the CAD tools, since one can quickly determine if the critical path is routed via direct, fast connections. As well, being able to see the structure of the critical path — for example, is it mostly short, local connections or mostly long connections — gives one insight into what kind of nets are determining the circuit speed and how the routing architecture could be changed to speed their routing.

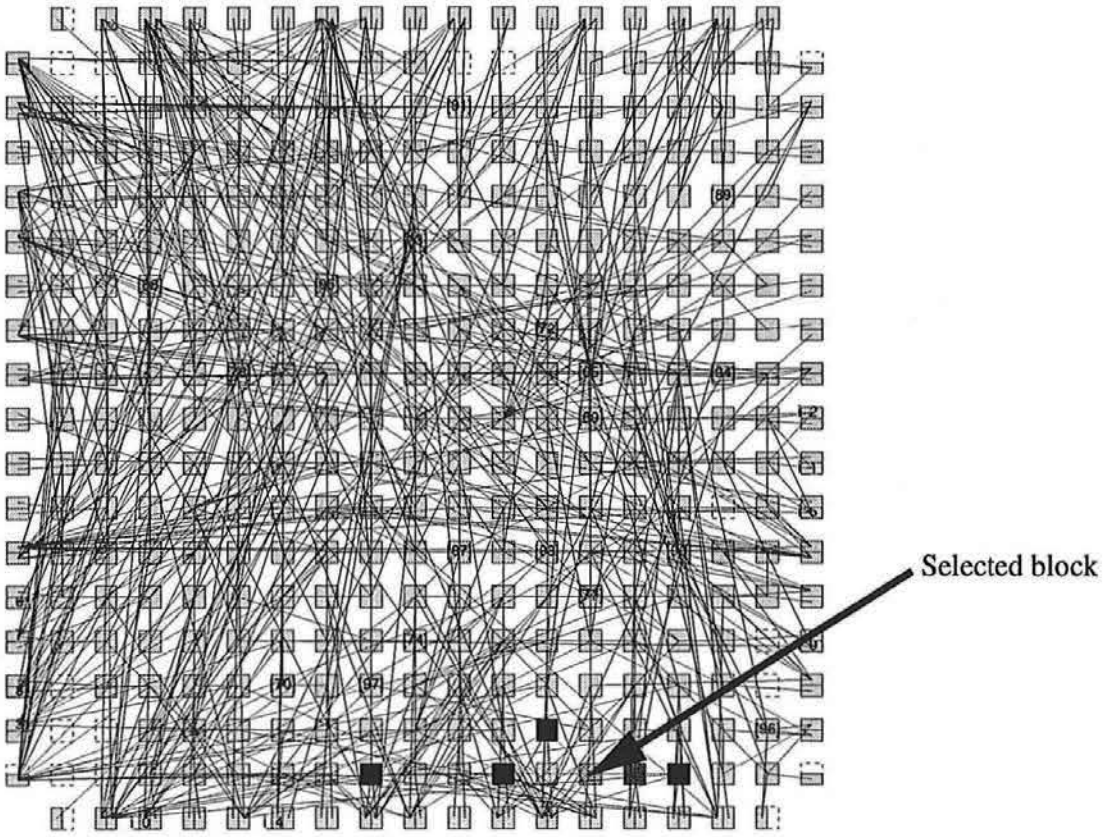


FIGURE A.2 Final placement of e64 found via simulated annealing.

When a routing fails because there are insufficient tracks, or because a switch pattern is simply unroutable, one can view all the routing resources that are still overused after the final iteration of the router. Showing where congestion is occurring in this way makes it much easier to determine why a circuit is not routing on a particular architecture — sometimes it is a simple matter of insufficient wiring, but other times there are intrinsic flaws in the architecture's switch pattern that make certain connection topologies difficult or impossible to route. The e64 circuit requires 10 tracks per channel to successfully route. Figure A.5 shows the congestion remaining after the last (thirtieth) router iteration when each channel contains only 8 tracks. Notice that both some wire segments and some logic block pins are overused.

VPR can also display the routing architecture of an FPGA. Every routing wire, logic block pin, and switch is displayed, and the data to be drawn is taken directly from the routing-resource graph. This allows a user to visualize an entire routing architecture, and quickly determine if he or she has set all the architecture parameters correctly in the architecture file. It also allows one to examine the quality of the switch patterns generated by the VPR architecture generator, and to quickly test and debug improvements to the architecture generator. Figure A.6 shows the entire routing fabric of the

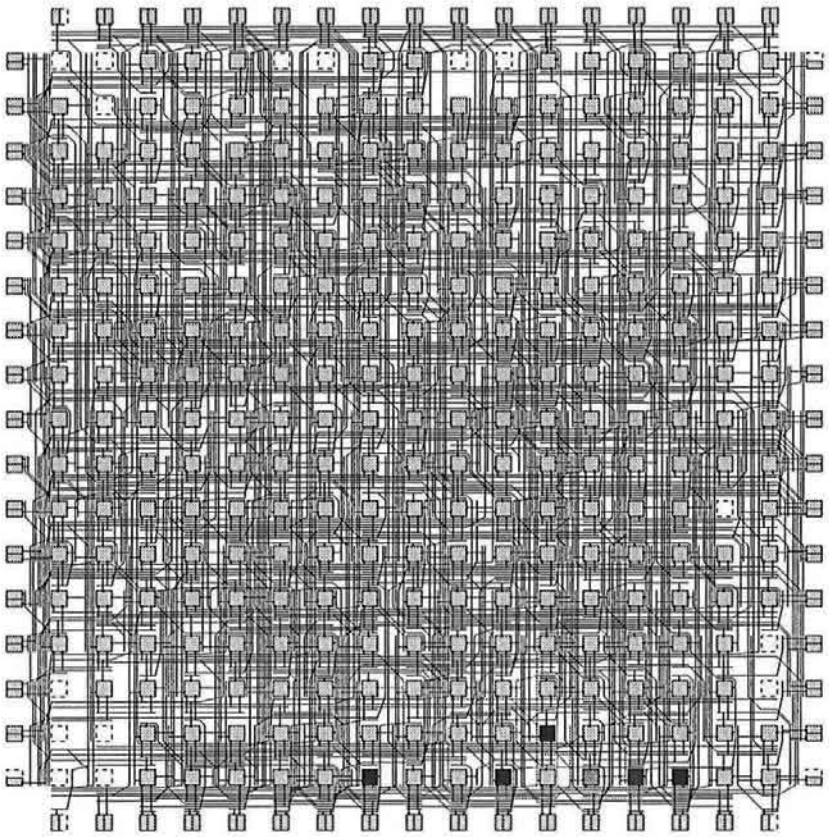
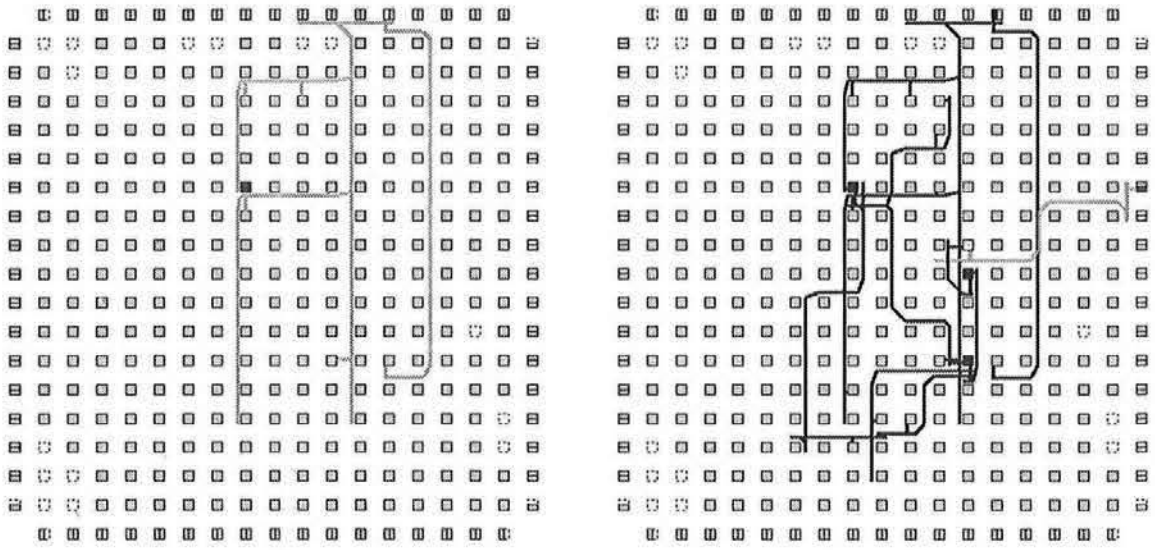


FIGURE A.3 Routing of benchmark circuit e64.



(a) (b)

FIGURE A.4 (a) First net and (b) all five nets on the critical path of e64.

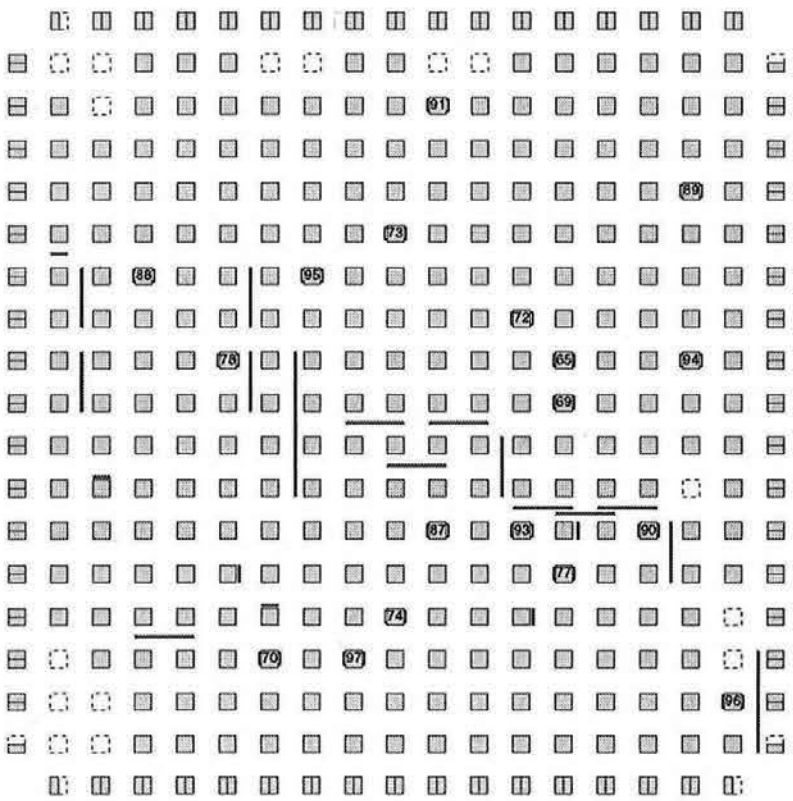


FIGURE A.5 Congestion remaining after an unsuccessful attempt at routing e64.

FPGA in which the e64 circuit was implemented in the previous figures. Figure A.7 is a close-up of a small part of the FPGA so the switch pattern is visible. The black lines are routing wires, while the small black squares are logic block input and output pins. Switches between logic block pins and routing wires are shown as x's. The routing switches in switch blocks are shown as grey lines between routing wires; a small triangle indicates the switch is a tri-state buffer, while a circle indicates it is a pass transistor.

Figure A.8 is a simpler view in which the routing switches are not displayed. The segmentation distribution of the FPGA is clearly visible: 20% of the routing tracks are length 1 wires, 40% are length 2 wires, and 40% are length 4 wires.

Finally, Figure A.9 shows a small portion of an FPGA with a more complex logic block. In Figure A.9 the logic block is a logic cluster of size 4, with 10 logic inputs, and one clock input. Notice that there are no switches connecting the clock input, which is in the upper right corner of each logic block, to routing wires, since in this architecture it was specified that the clock is routed on a special dedicated resource.

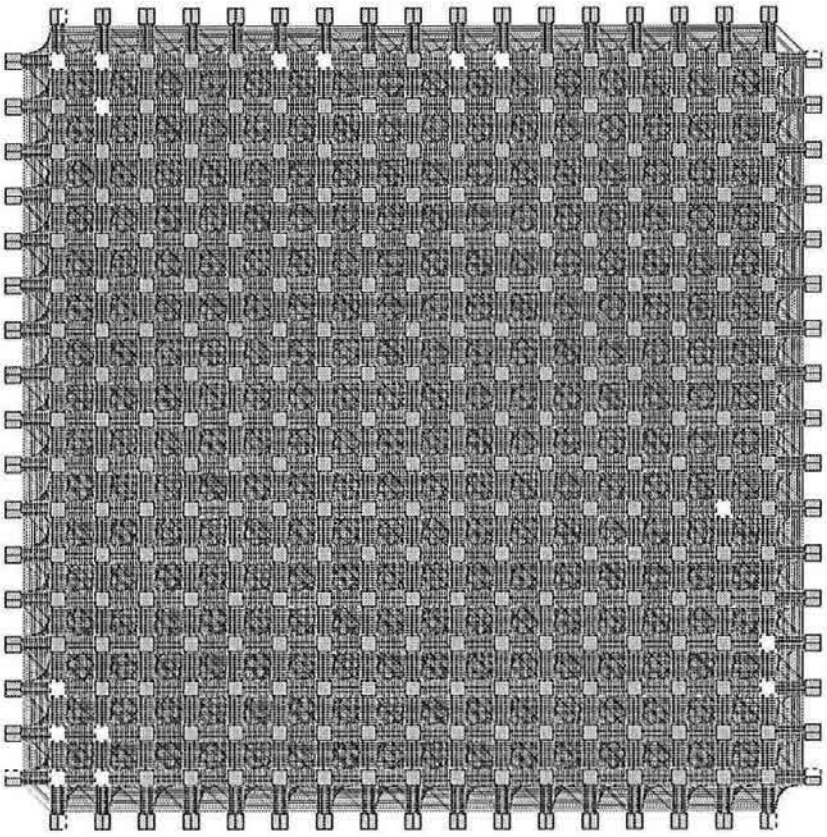


FIGURE A.6 Routing architecture of an FPGA with a single BLE logic block.

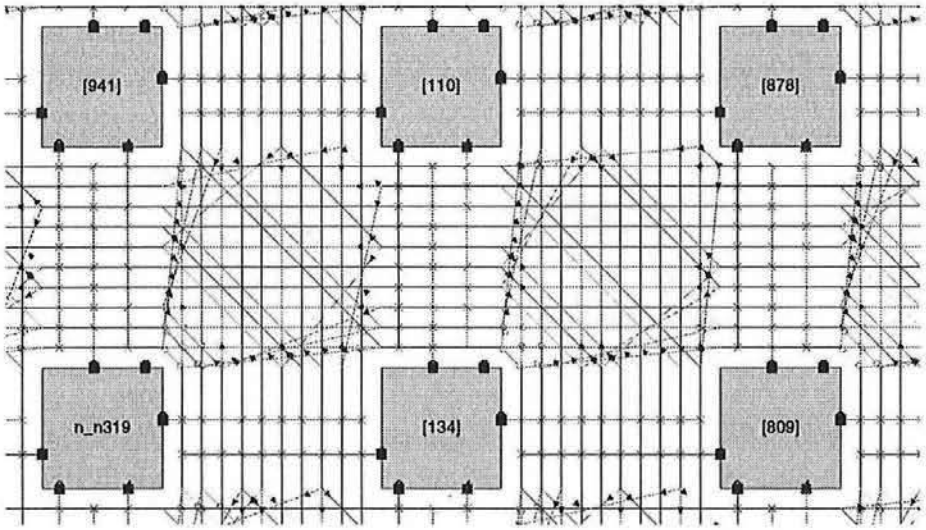


FIGURE A.7 Close-up of routing architecture of an FPGA with a single BLE logic block.

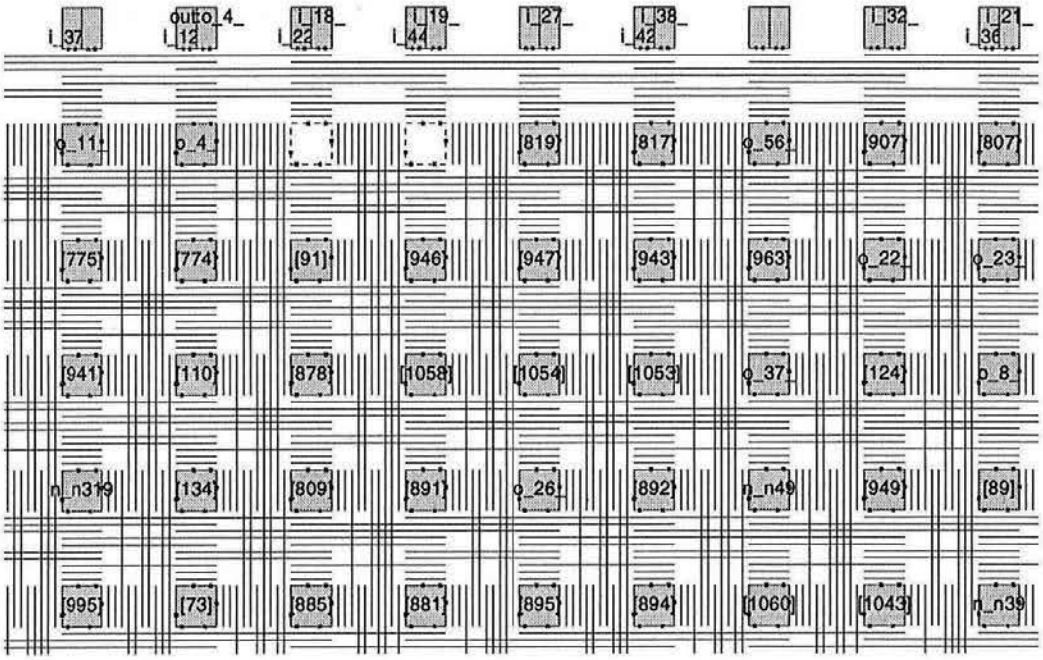


FIGURE A.8 Segmentation distribution of an example FPGA.

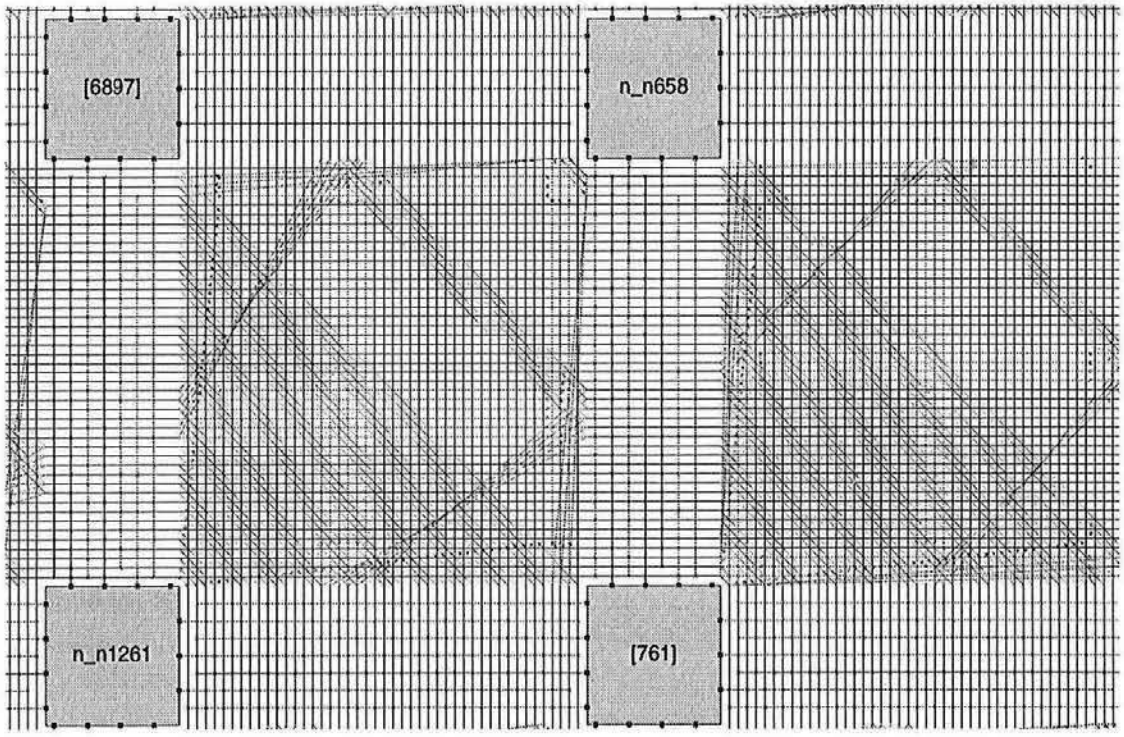


FIGURE A.9 An FPGA where each logic block is a size 4 logic cluster with 10 logic inputs and one clock input.

APPENDIX B

FPGA Circuitry and Process Modeling

This appendix provides more detail on how the area and delay of the FPGAs studied in Chapters 6 and 7 was determined. The next section provides transistor-level schematics showing how each structure in an FPGA is implemented. Most of these schematics are process-independent, but occasionally transistor sizing issues that are somewhat specific to a specific process arise. In such cases, we size the transistors assuming the FPGA would be implemented in TSMC's 0.35 μm, 3.3. V process [154]. Section B.2 describes how we determined the delay or equivalent RC circuit for each structure in an FPGA, and lists the delay through some of the key FPGA structures. This data is required to apply the FPGA delay model described in Section 6.2.3.

B.1 Transistor-Level Schematics and Assumptions

In this section we detail the transistor-level schematics for each FPGA structure used in this research. These schematics show how many transistors each structure contains, and how large the various transistors are, which is exactly the information required by the area model described in Section 6.2.2. As well, these transistor-level schematics are the structures simulated to extract the delay values used in Chapters 6 and 7. Chow et al [21] contains another discussion of the transistor-level implementation of an FPGA; most of the structures we use are similar.

B.1.1 FPGA Routing Structures

A key circuit element of an SRAM-based FPGA is the SRAM cell itself. We used the 6-transistor, double-ended programming cell illustrated in Figure B.1. Although a five-transistor SRAM cell is possible, a six-transistor cell is generally more stable — i.e. more resistant to state flipping due to crosstalk or charge sharing. While some of these transistors may have slightly greater than minimum width (or some may have more than the minimum channel length of 0.35 μm), we assume the total structure requires 6 minimum-width transistor areas. Since the layout of an SRAM-cell is usually the most heavily optimized layout in an FPGA (often to the point that special, relaxed design rules are used in its layout), the assumption that it requires only 6 minimum-width transistor areas is reasonable. We assume that the data and $\overline{\text{data}}$ values stored in the cell can be connected directly to other wires without any isolating inverters — the stability of a six-transistor SRAM cell is therefore essential.

Figure B.2(a) shows a pass transistor used in an FPGA's routing. Each routing pass transistor is controlled by a single SRAM cell. Note that usually this pass transistor will be considerably larger than minimum-width; Appendix C.1 studies the question of how best to size this transistor. If the width of this transistor is ten times the minimum, which is the size we use in Chapter 7, the number of minimum-width transistor areas required by a pass transistor routing switch is 6 (SRAM cell) + 5.5 (ten times minimum size pass transistor; see Equation (6.1)) for a total of 11.5.

Gate Boosting

Figure B.2(b) shows a potential problem with the use of nMOS pass transistors. In Figure B.2(b), an "on" routing pass transistor is passing a logic high value to a routing wire. Notice that the voltage of the routing wire is less than V_{dd} (3.3 V), since an nMOS pass transistor degrades a logic high value by a threshold voltage (including any increase in the nominal threshold voltage due to the body effect [155]). In

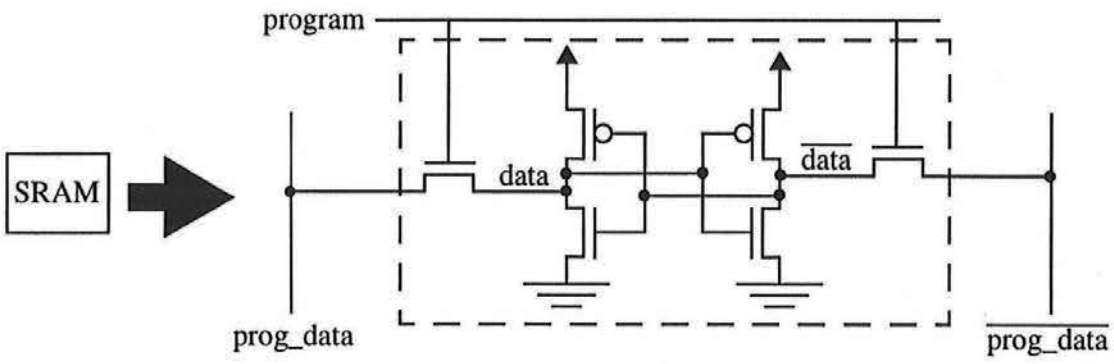
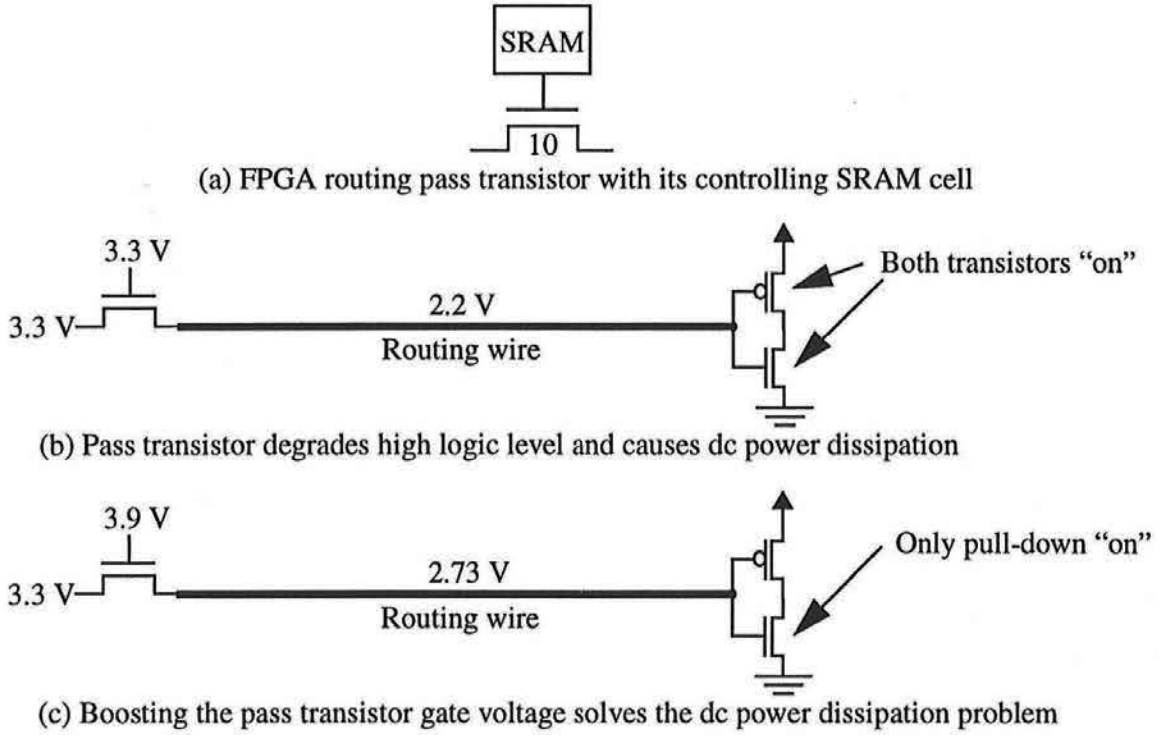


FIGURE B.1 Six-transistor SRAM cell schematic.



(a) FPGA routing pass transistor with its controlling SRAM cell

(b) Pass transistor degrades high logic level and causes dc power dissipation

(c) Boosting the pass transistor gate voltage solves the dc power dissipation problem

FIGURE B.2 (a) FPGA routing pass transistor; (b) static power problem; and (c) solution.

TSMC's 0.35 μm process, for example, the routing wire voltage will rise only to 2.2 V. The output voltage is degraded by 1.1 V, even though the nominal threshold voltage in this process is only 0.6 V, because the pass transistor suffers from the body effect. Downstream, a buffer is used to sense and re-power the signal on the routing wire. With an input voltage of only 2.2 V, both the pMOS and nMOS transistors in the first stage of this buffer will be at least partially "on", dissipating unacceptable amounts of DC power. For example, an FPGA large enough to accommodate the clma benchmark circuit (which requires 2121 size 4 logic clusters) would dissipate 21 W of static power in these partially "on" inverters alone.

One solution to this problem is to boost the gate voltage of pass transistors above the power supply; this technique has been used by Xilinx in their FPGAs. We assume that the gates of all the pass transistors in the FPGAs we construct are boosted to 3.9 V (one nominal threshold voltage above V_{dd}) when designing FPGA circuitry or extracting circuit delays. Figure B.2(c) shows that with a gate boosted to 3.9 V, the pass transistor will now pass a logic high level of 2.73 V, which is enough to turn off the pMOS pull-up transistor in any downstream buffer. For an FPGA large enough to accommodate the clma benchmark circuit, this gate boosting technique reduces the static power dissipated in these "downstream" buffers from 21 W to 0.041 W.

Buffers

Multi-stage buffers are heavily used in the design of FPGAs. To drive a large load, one usually builds a chain of inverters of gradually increasing size — a multi-stage buffer. We build buffer chains with a stage ratio of as close to 4 as possible; this yields high speed while minimizing area [155]. In these buffer chains the pull-up pMOS transistors are 1.9 times as wide as the nMOS transistors. This is roughly the square root of the ratio between the drive strength of an nMOS and the drive strength of a pMOS transistor of the same size in TSMC's 0.35 μm process. Sizing the pMOS transistors in this way minimizes the delay through the buffer chain [156]. Even more importantly, using a pMOS to nMOS width ratio of 1.9 results in equal rise and fall times for the tri-state buffer structures discussed below. Figure B.3 shows an example buffer chain used to create an output stage with 16 times the drive strength of a minimum-size buffer. Note that all the transistor width labels in Figure B.3 (and the other figures in this appendix) are "times minimum width" values, rather than widths in μm.

Recall that signals generated by nMOS pass transistors swing only from 0 V to 2.73 V (rather than 3.3 V). To achieve equal rise and fall times, the inverter that senses the state of a wire driven by an nMOS pass transistor uses a pMOS to nMOS width ratio of only 1.7, rather than the 1.9 we usually use. This reduces the switching threshold of the inverter to 1.35 V (about midway between 0 and 2.73 V), and yields equal rise and fall times on the nMOS pass transistor output.

The buffered switches in FPGA switch blocks and the switches between logic block output pins and routing tracks (in the "output pin" connection blocks) are tri-state buffers. There are several ways to construct tri-state buffers; Figure B.4 shows the form of tri-state buffer we assume. It is simply a combination of a (potentially multi-stage) buffer with an nMOS pass transistor. We assume that the gate of this pass transistor (like all the other pass transistors in the FPGA) is boosted to 3.9 V. The drive strength of the tri-state buffer in Figure B.4 is five times the minimum, which is the tri-state buffer size used in Chapter 7. It occupies 19.7 minimum-width transistor areas.

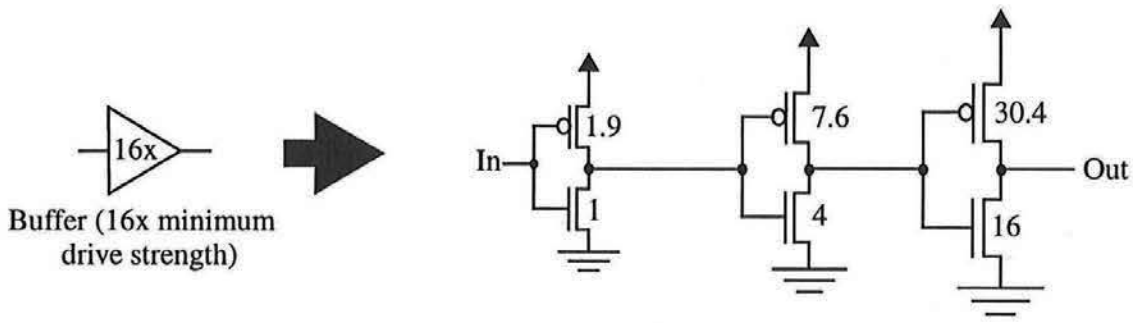


FIGURE B.3 Schematic of a multi-stage buffer.

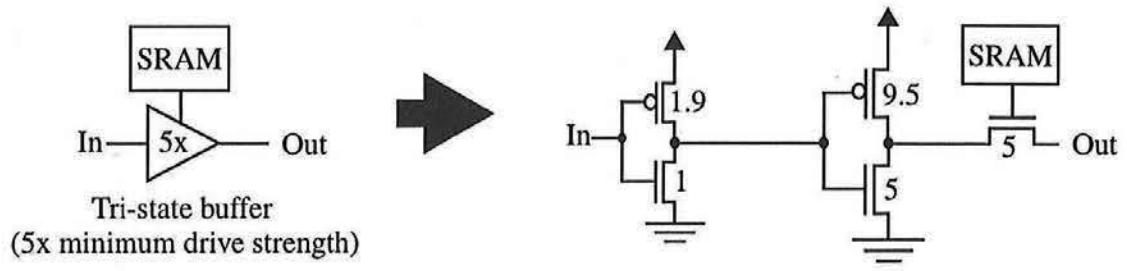


FIGURE B.4 Schematic of a tri-state buffer with 5 times minimum drive strength.

At a point where there are multiple buffered routing switches with the same input, there are two ways to implement the group of switches. We can either build each switch as a completely separate tri-state buffer, as Figure B.5(a) shows, or we can build only one buffer chain, with several pass transistors connecting to the output of this buffer, as shown in Figure B.5(b). We call the two implementations "no buffer sharing" and "fan-out buffer sharing," respectively. Implementing only one buffer chain (i.e., fan-out buffer sharing) saves a significant amount of area, but it can degrade speed by a small amount since this buffer will be more heavily loaded if several of the pass transistors connecting to its output are "on."

The area model built into VPR outputs two area estimates: one assuming no buffer sharing, and the other assuming fan-out buffer sharing. The no-buffer-sharing area assumes that each tri-state buffer switch in a switch block, or in an output pin connection block, is implemented as a separate buffer chain plus pass transistor, as Figure B.5(a) shows. This area estimate is pessimistic because significant area savings can be achieved by sharing one buffer chain. The fan-out-buffer-sharing area model assumes instead that only one buffer chain is implemented for each logic block output pin, or set of tri-state buffer routing switches with the same input and at the same switch block. This area model is depicted in Figure B.5(b). We call this area model slightly optimistic for two reasons. First, in a real FPGA it is likely that the size of the buffer chain would be increased somewhat as the number of pass transistors connected to its output increased, in order to minimize any speed degradation due to several of the pass transistors being "on" at once. Hence the fan-out-buffer-sharing area model slightly underpredicts routing area. Second, the delay model in VPR assumes each switch is independent — that is, does not share a buffer chain with another switch. Consequently, VPR's delay calculator does not account for the speed reduction caused by several of the pass transistors connected to one buffer chain being "on." In other words, the delay model built into VPR models the no-buffer-sharing case. The difference in delay caused by sharing buffers is not very significant, however, so we consider the fan-out-buffer-sharing area model the more realistic one.

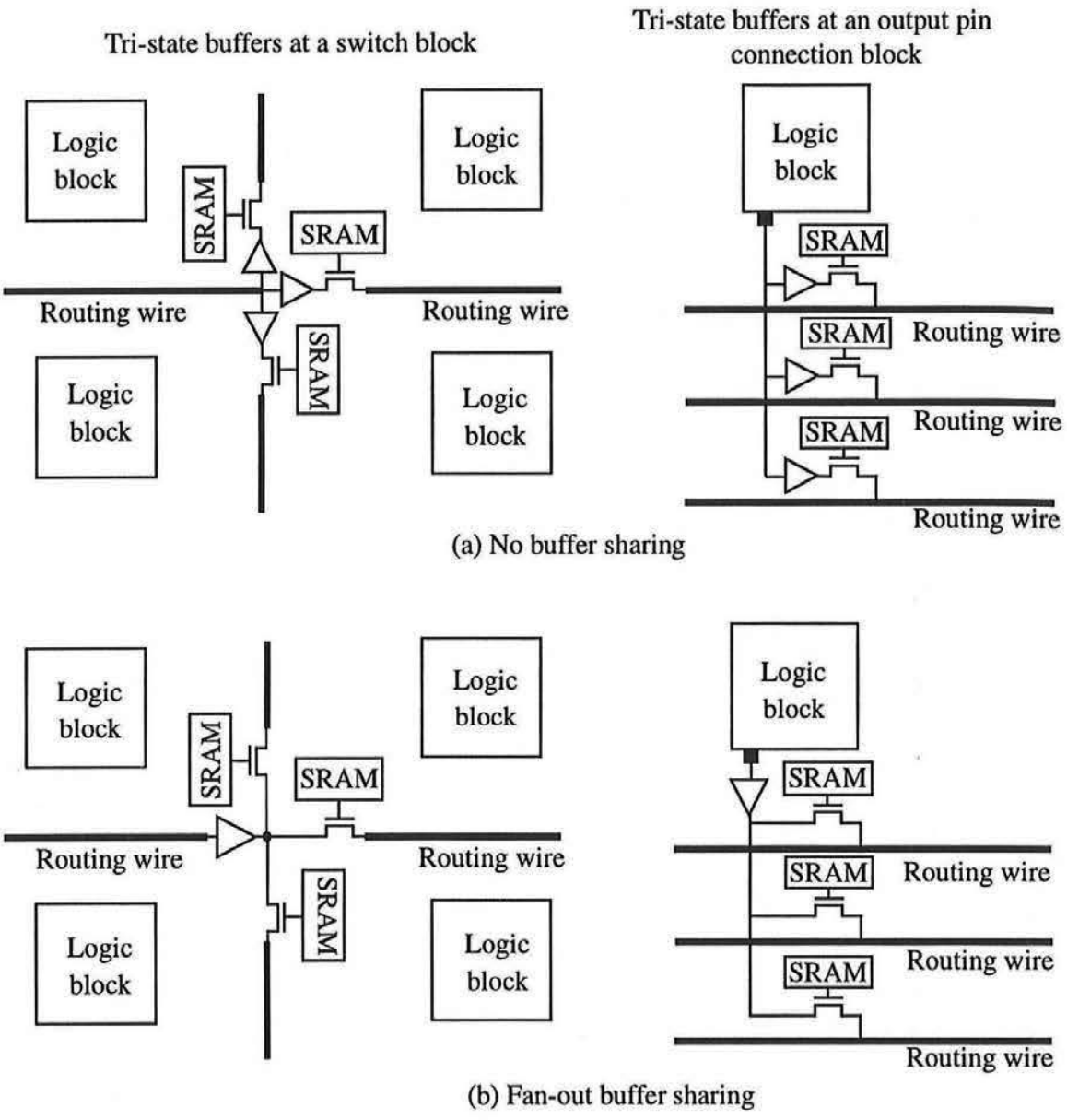


FIGURE B.5 Methods to build tri-state buffers at switch blocks and output pin connection blocks.

Accordingly, all the area results in Chapters 6 and 7 were determined with the fan-out-buffer-sharing area model.

Connection Block to Logic Block Input Pins

The connection block from a set of routing tracks to a logic block input pin is implemented as a multiplexer, as shown in Figure B.6. Notice that buffers isolate each track from the input capacitance of the connection blocks attaching to it. We assume

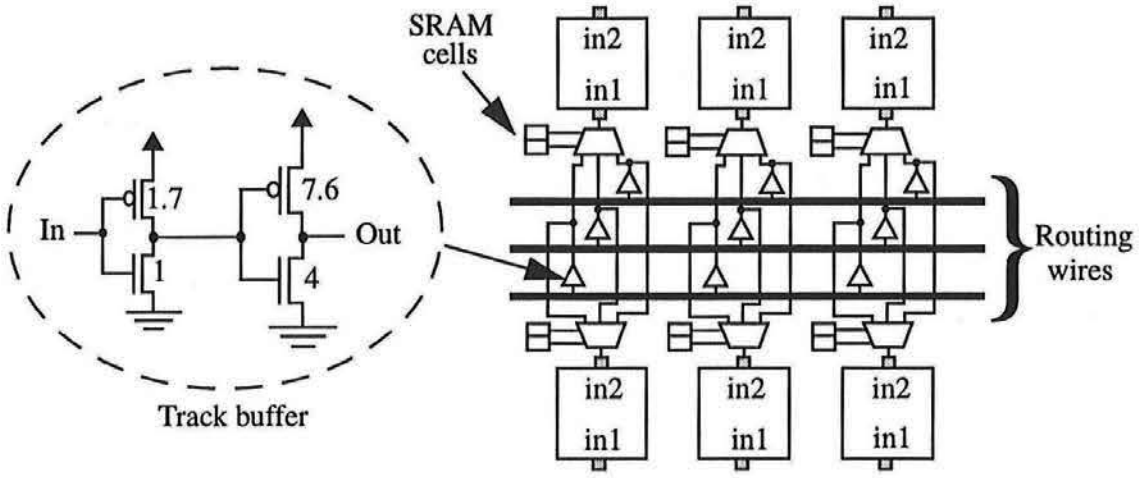


FIGURE B.6 Implementation of input pin connection blocks.

only one buffer is needed at each logic block location along a wire — a wire of length 4, for example, has four attached "track buffers." The multiplexers attaching to a wire from the logic block above a certain point on the wire and the logic block below that point on the wire connect to the output of one track buffer. Figure B.6 also shows that we assume the track buffer is a two-stage buffer with the second stage having a drive strength of four times the minimum. Notice that the first stage of the track buffer uses a pMOS to nMOS width ratio of 1.7 to achieve equal rising and falling delays on the routing wires.

Figure B.7 shows that multiplexers are implemented as a binary tree of pass transistors. All the transistors in this tree are minimum-width. For input pin connection blocks, SRAM cells control which multiplexer input is selected. The total number of minimum-width transistor areas required by such a multiplexer with N_{inputs} inputs is:

$$Area(N_{inputs}) = 6\lceil \log_2(N_{inputs}) \rceil + 2N_{inputs} - 2 . \quad \textbf{(B.1)}$$

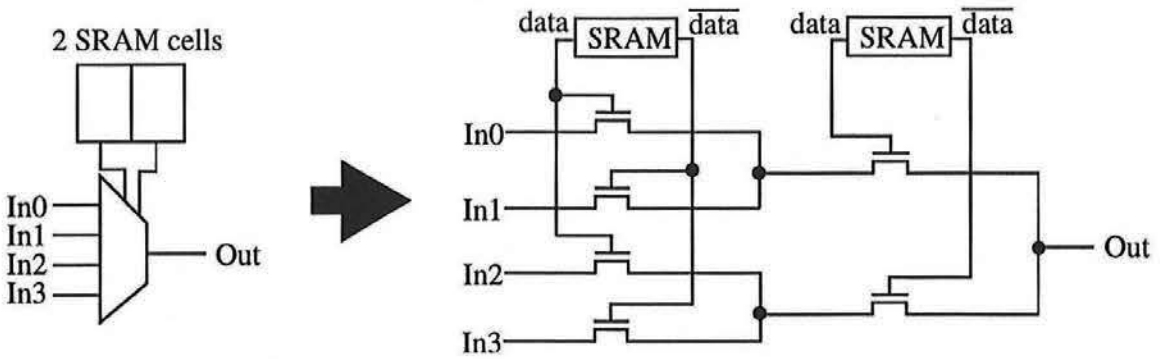


FIGURE B.7 Schematic of a 4-input SRAM-controlled multiplexer.

Since we are implementing multiplexers as pass transistor trees, we need buffers on the inputs and the output of the multiplexer. If these buffers are not already present in the circuit, they must be added and their area included in the total area.

B.1.2 Logic Block Structures

Figure B.8 shows the logic block used in Chapter 7: a logic cluster of size 4 with 10 logic inputs and one clock. We also assume there is one set/reset signal of programmable polarity shared by all four registers in the logic block. Notice that signals that have to travel a significant distance across the logic block or which have a large fan-out pass through two-stage buffers with a drive strength of two to four times the minimum to amplify them. The fan-out of some of these buffers is a function of the logic cluster size — our area model resizes these buffers appropriately for the fan-out present in each logic cluster. Note that to keep Figure B.8 understandable, we do not show all the repeated structures in a logic cluster of size 4. Specifically, we show only two of the ten input pin buffers, two of the sixteen 14:1 multiplexers, and one each of the four 4-LUTs, registers, and BLE output buffers contained in a size 4 logic cluster.

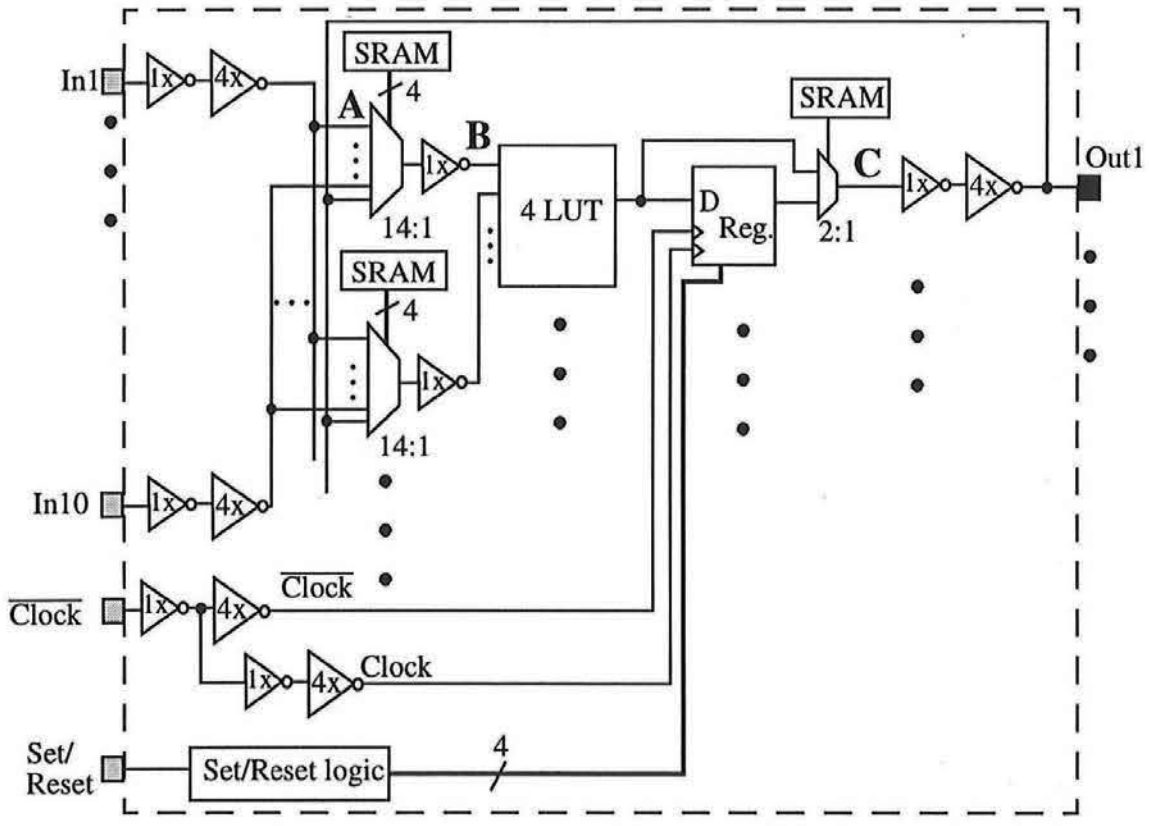


FIGURE B.8 Schematic of a logic cluster of size 4 with 10 logic inputs, 1 clock and a set/reset input.

As well, note that the four times minimum strength buffer driving the output pin is strong enough to drive the local routing, but not, in general, the routing tracks. Appendix C.3 discusses the size of output buffer we use to drive different types of routing wires. The **A**, **B**, and **C** points labelled in Figure B.8 will be referred to in Section B.2, where we summarize the delay of the various paths in this logic block.

Figure B.9 shows our implementation of a 4-input look-up table. It uses sixteen SRAM cells, a sixteen-input pass transistor multiplexer, and a set of buffers to amplify the select inputs and generate their complements. Note that only 6 of the 16

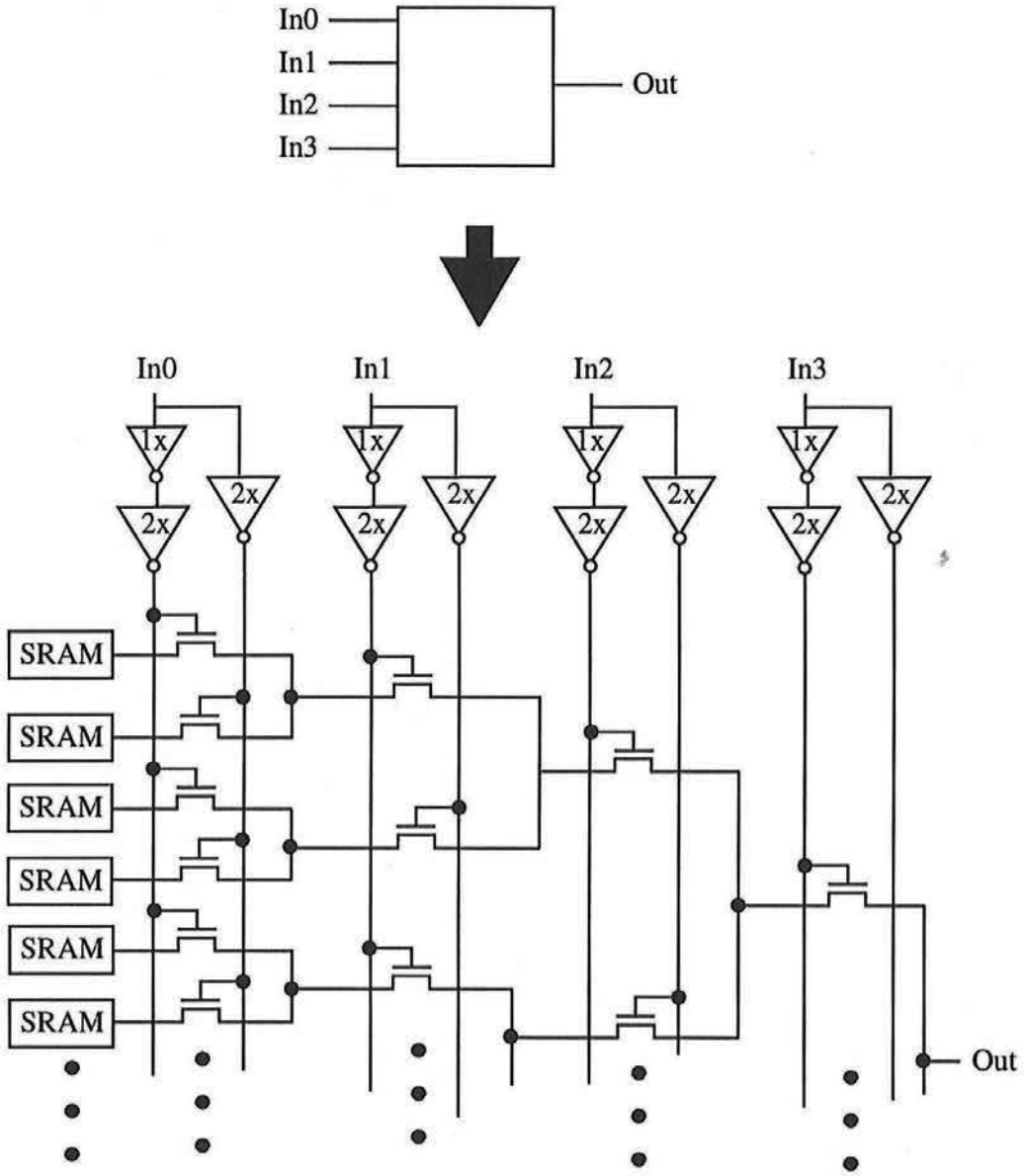


FIGURE B.9 Schematic of a 4-input look-up table.

SRAM cells are shown, and only a portion of the pass transistor multiplexer tree is shown. The area of this 4-LUT is 96 (SRAM cells) + 30 (multiplexer tree) + 41 (input buffers and complementers) = 167 minimum-width transistor areas.

Figure B.10 shows the master-slave register with asynchronous set and reset inputs that we use. Numbers given after a slash (/) denote the channel length (in μm) of transistors that use a longer channel than the minimum 0.35 μm. This register contains 16 transistors, but some are larger than minimum width and some have greater than the minimum channel length, so it occupies 19 minimum-width transistor areas.

Finally, Figure B.11 shows the set/reset logic circuit used to enable programming of both the polarity of the set/reset input to a logic block, and whether this signal sets or resets the registers in a logic block. This circuitry occupies 48.5 minimum-width transistor areas.

Summing the areas of all the structures in the size four logic cluster of Figure B.8 yields the total area: 2·9.25 (clock buffers) + 16·50 (local routing multiplexers) + 16·2.35 (buffers on outputs of local routing multiplexers) + 4·167 (four 4-LUTs) + 4·19 (four registers) + 4·8 (four 2:1 multiplexers) + 48.5 (set/reset logic) = 1678 minimum-width transistor areas for this logic block. The highly attentive reader will notice that this summation does not include the ten buffers on the logic block input pins, nor the four buffers on the logic block output pins. The area of these buffers is counted as part of the routing area by VPR.

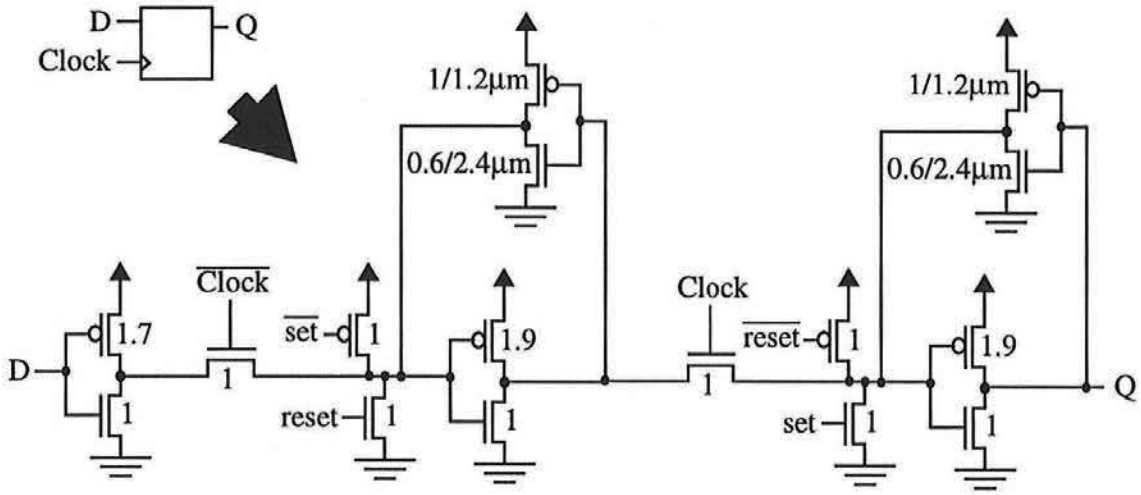


FIGURE B.10 Master-slave register with asynchronous set and reset.

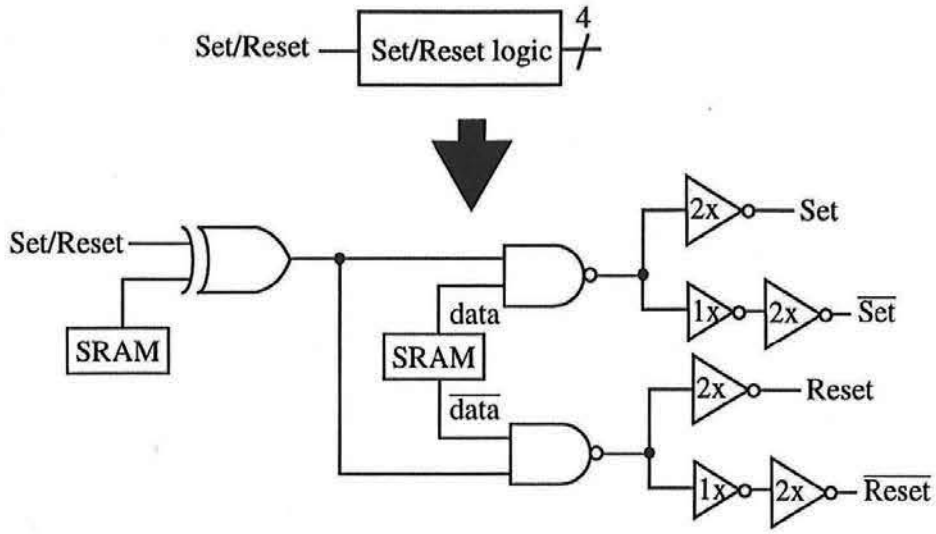


FIGURE B.11 Circuitry to generate and amplify the set and reset signals within a logic block.

B.2 Delay and RC-Equivalent Circuit Extraction

In this section, we describe how we determined the delay or equivalent RC circuit for each FPGA structure, and we list the delay through some of the key FPGA structures. This delay and RC data is required to apply the FPGA delay model described in Section 6.2.3.

We first performed SPICE simulations to determine the delay of all the paths in the logic block shown in Figure B.8, as well as the delay from a routing track to a logic block input pin. The delay of a path is the difference between the time at which the path input signal passes through the midpoint of its voltage swing and the time at which the output signal passes through the midpoint of its voltage swing. The results are listed in Table B.1. The letters in Table B.1 refer to the points labelled on Figure B.8.

We performed SPICE simulations to determine the intrinsic delay of the (often multi-stage) buffers we use in the routing. The buffer intrinsic delay is the delay, when the buffer is unloaded, between the buffer input passing through its 50% swing voltage and the buffer output passing through its 50% swing voltage. Because FPGA routing switches are heavily loaded, the slew rate of the voltage input signal to a routing buffer is fairly low. This slowly changing input affects the delay through the buffers — we found that the buffer delay is 100 ps higher for buffers in the FPGA routing

TABLE B.1 Cluster-based logic block of size 4 delay values in TSMC's 0.35 µm CMOS process.

Path	Passes through	Delay (ps)
Routing track → Point **A**	Track buffer, connection block multiplexer, input pin buffers	1040
Point **A** → Point **B**	14:1 multiplexer, buffer	395
Point **B** → Point **C**	4-LUT, 2:1 multiplexer	465
Point **C** → Point **A**	Local routing buffers	280
Point **B** → Register storage node	4-LUT, register set-up time	670
Register storage node → Point **C**	Register clock-to-Q delay, 2:1 multiplexer	332

than it is for buffers in the FPGA logic block because of the difference in input slew rates. We do not provide the buffer intrinsic delays here because it would violate our non-disclosure agreement with TSMC.

We found the equivalent resistance of nMOS pass transistors via the circuit shown in Figure B.12. The Elmore delay of this circuit is simply $R_{eq,n} \cdot C$. Since our logic block and buffer intrinsic delays all give the time required for 50% signal swing, we want the Elmore delay of the circuit in Figure B.12 to give us the 50% signal swing time as well. We accomplish this simply by setting $R_{eq,n} \cdot C$ equal to the 50% swing time measured from SPICE, and solving for $R_{eq,n}$. We found that the rise and fall times of signals generated by pass transistors were essentially equal, so long as we measure the time for the signal to change to 50% of its final value, and not to $V_{dd}/2$. By properly sizing the n and p transistors in the buffer used to sense the state of this routing wire (the dotted buffer in Figure B.12), we can adjust the "switch point volt-

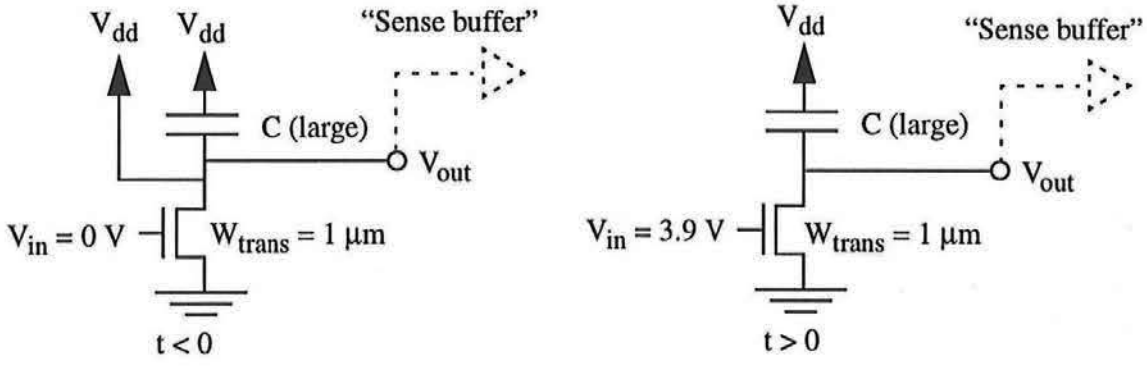


FIGURE B.12 Circuit used to determine equivalent resistance of nMOS transistors.

age" to lie at this "signal swing midpoint" voltage. Note that the drive strength of CMOS transistors increases linearly with transistor width, so the equivalent resistance of transistors with widths different than 1 μm is simply R_{eq} / W_{trans}. A similar procedure was used to find the equivalent output resistance of pMOS transistors and tri-state buffers. Again, our nondisclosure agreement precludes the listing of these equivalent resistances here.

To determine the metal resistance and capacitance of routing wires, we need to know how long these wires are. In other words, we need to know the size of a basic FPGA tile. For example, consider an FPGA employing a size 4 logic cluster as its logic block. We determined the basic tile size by examining a representative routing architecture (50% length 4 buffered wires, 50% length 4 pass-transistor-switched wires) as follows:

1. The most difficult to route benchmark circuit, pdc, requires 62 tracks per channel in this architecture; hence we set the channel width to 62.
2. The number of minimum-width transistor areas in the routing for this architecture and a channel width of 62 is 6857. Adding in the 1678 minimum-width transistor areas in the logic block yields a total of 8535.
3. We assume that a good layout team can achieve 60% of the maximum achievable transistor density. Therefore:

$$Area(\mu m^2) = MinWidthTransAreas \cdot \frac{6.2275\ \mu m^2 / \text{MinWidthTrans}}{0.6} = 88\,600\ \mu m^2 \quad \textbf{(B.2)}$$

Assuming a square logic block, then, we obtain a tile size of 300 μm on a side; we assume this is the tile size for each architecture examined in Chapter 7. To check that this value is reasonable, we also measured the area per logic block of a Xilinx XC4085XL FPGA (in a 0.35 μm process) and found that it is 115 000 μm^2, which leads to an (assumed square) tile size of 340 μm x 340 μm. The area of an Altera Flex10K100 tile is 259 000 μm^2 in a 0.5 μm^2 process; scaling to a 0.35 μm process and dividing by two (since each Altera logic block contains 8 4-LUTs, rather than 4), yields an area per four 4-LUT tile of 129 600 μm^2, or 360 μm x 360 μm in a square layout. Consequently, our estimate of a tile size of 300 μm on a side for a logic block containing four 4-LUTs looks quite reasonable.

In modelling metal resistance, we want $R_{metal,eq} \cdot C$ to give the 50% signal swing delay so that it is compatible with our other delay values. Since metal resistance is a true linear resistor, we know that the voltage on a capacitor being charged by resistive metal will follow an exponential curve in time. The time constant of this exponential is $R_{metal} \cdot C$, where R_{metal} is the "true" metal resistance given in the foundry data sheets. Consequently, we know that the 50% signal swing delay will occur at:

$$t_{50\%\,\text{swing}} = \ln(0.5) \cdot R_{metal} \cdot C = 0.693 \cdot R_{metal} \cdot C\,. \tag{B.3}$$

From (B.3) it is clear that $R_{metal,eq}$ is $0.693 \cdot R_{metal}$.

The metal capacitance data can be taken directly from the foundry data sheets. We extracted the transistor gate capacitance, and the pMOS and nMOS diffusion capacitances, from the foundry SPICE models. Since the diffusion capacitances are bias voltage dependent, we obtained the average diffusion capacitances over the entire bias range via:

$$C_{\text{diff,av}} = \frac{1}{3.3} \cdot \int_{0}^{3.3} C_{\text{diff}}(V)dV \tag{B.4}$$

and used this average diffusion capacitance value in our RC equivalent circuit model, as suggested in [156].

Finally, recall from Section B.1.1 that we are boosting the gates of pass transistors to 3.9 V in order to control the FPGA static power dissipation. It is interesting to note that this gate boosting does not change the routing delay very much — the transistor drive strength increases, but so does the signal swing, and the two effects largely cancel each other.

APPENDIX C

Sizing of Routing Transistors and Metal

Both the area and delay of an FPGA depend strongly on the size of the routing transistors and the metal width of the routing wires. In this appendix we show how we determined appropriate sizes for the pass transistors and tri-state buffers used in FPGA routing, and the best metal width for routing wires, for use with a size 4 logic cluster. All the delay values in this section were determined from the switch-level (resistors, capacitors and ideal delay elements) delay model we extracted from TSMC's 0.35 μm process; this delay modelling procedure was described in Appendix B.2. Throughout this section we also assume that the layout area of a basic FPGA tile (in this process) is 300 μm x 300 μm, as described in Appendix B.2. A size 4 logic cluster results in a basic tile of this size; for logic clusters of different sizes the tile size, and hence the best routing transistor sizes, will change.

C.1 Sizing Pass Transistor Routing Switches

In this section, we determine the best routing pass transistor width by evaluating the effect of different pass transistor widths on both the speed of the routing wire shown in Figure C.1, and the area of the routing switches. While Figure C.1 shows a wire of length 4, we will vary the length (L_{wire}) of this routing wire over a wide range in this section. We assume the wire is fully switch-block and connection-block populated, and that the switch block topology used leads to an F_s of 3 at the wire ends, and an F_s value of 1 at the "internal" switch block points. The disjoint switch block topology used in Chapters 6 and 7 matches these F_s values. We also assume there is a pass transistor allowing this wire to connect to a logic block output pin at each of the L_{wire}

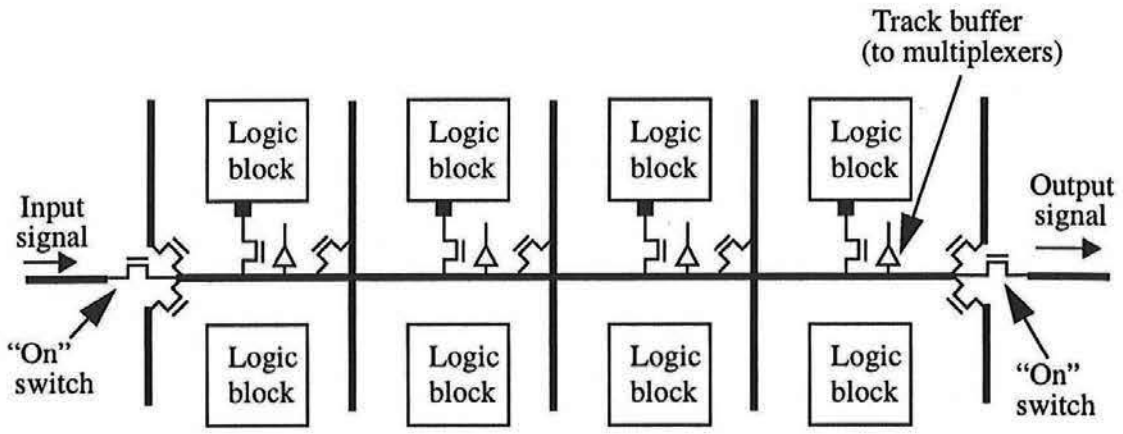


FIGURE C.1 Wiring topology assumed in pass transistor sizing experiments.

logic blocks spanned by the wire. These transistors connecting the wire to logic block output pins are the same size as the other routing transistors. A track buffer with a small first stage (as described in Appendix B.1.1) also loads the wire at each logic block it passes. Finally, in this section we assume that the routing wire is laid out in metal 3 with minimum width and minimum spacing. We are interested in determining the delay of the wire in Figure C.1 when one pass transistor at each end is "on" to create a multi-wire connection — all the other switches are off and only add capacitive loading.

The Elmore delay of a connection that passes through M pass transistor switches and routing wires is:

$$T_{Elmore}(M) = M \cdot \left[\frac{(M+1)}{2}(R_{pass} + R_{metal})C_{total} - \frac{R_{metal}C_{total}}{2}\right] \quad \textbf{(C.1)}$$

where C_{total} is the total (metal + transistor parasitics) capacitance loading the wire, R_{pass} is the equivalent resistance of a pass transistor, and R_{metal} is the end-to-end resistance of the wire. The dominant (quadratic in the number of routing switches in series) delay term is proportional to:

$$\text{Dominant delay constant} = (R_{pass} + R_{metal})C_{total}. \quad \textbf{(C.2)}$$

Figure C.2 plots this dominant delay constant as a function of the pass transistor width for several different wire lengths. The delay constant of each wire has been divided by the wire length, L_{wire}, to allow all the curves to be plotted on the same axes — this does not affect the shape of the curves in any way. The horizontal axis in Figure C.2 is the width, relative to the minimum contacted transistor width of 0.7 μm, of the routing pass transistors. As the width of the pass transistors increases, the wir-

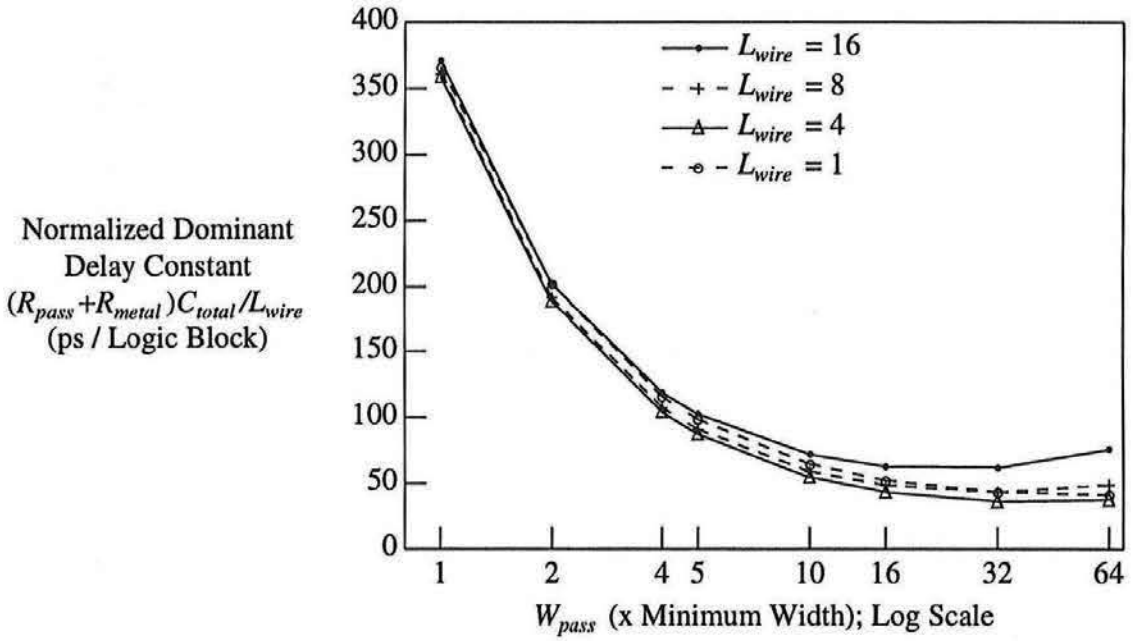


FIGURE C.2 Normalized dominant delay constant vs. routing pass transistor width.

ing delay clearly drops significantly. For very large pass transistors the delay stops improving, since the switch capacitances become larger than the metal capacitance and the wire becomes self-loaded. For the longer wires, delay actually degrades once the transistor width is more than about 30 times minimum, since the metal resistance begins to become more significant than the pass transistor resistance.

Figure C.3 shows the area required by each pass transistor routing switch, including the area of the controlling SRAM cell. Obviously the area of each routing switch increases as the pass transistor size is increased. The question, then, is how best to size the routing pass transistors to achieve the best area-delay trade-off. Intuitively, we want to choose the transistor width that is nearest the "knees" of both the delay constant and switch area curves. A more quantitative way of formulating this goal is to size routing transistors to minimize the area-delay product of the resulting FPGA. As described in Section 6.2.4 minimizing the area-delay product of an FPGA maximizes the FPGA's computational throughput (for parallel algorithms).

To find the FPGA with the minimum area-delay product achievable with a given logic block and routing architecture, we would have to simultaneously vary the size of every transistor in a basic tile of the FPGA, and determine the area and delay achieved by the FPGA under each transistor sizing. Clearly this is a dauntingly large and complex search space. We can, however, *approximately* minimize the area-delay product

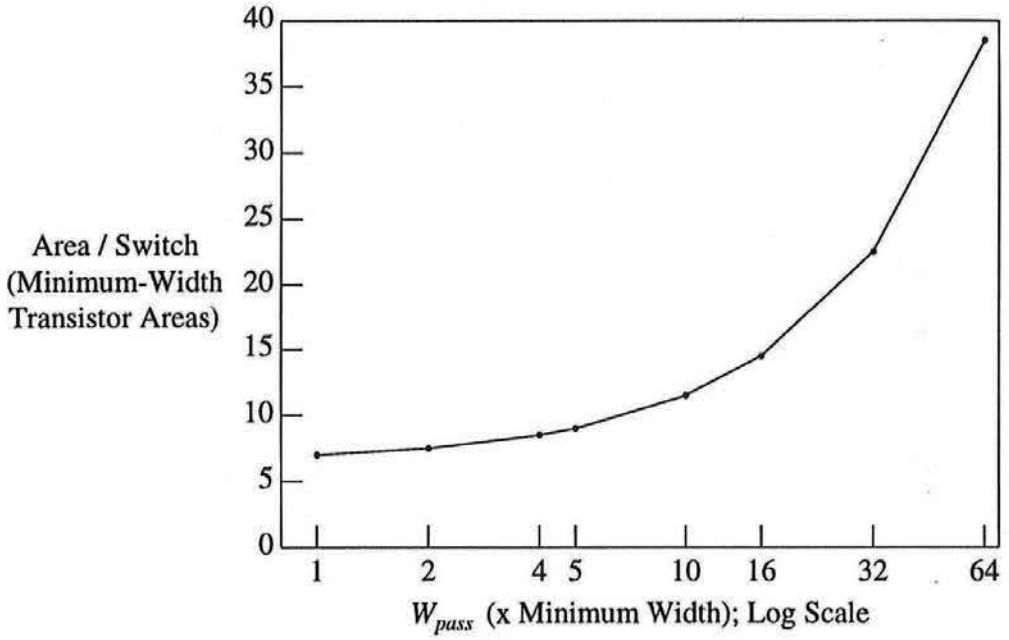


FIGURE C.3 Area of a pass transistor routing switch (including the controlling SRAM bit) vs. pass transistor width.

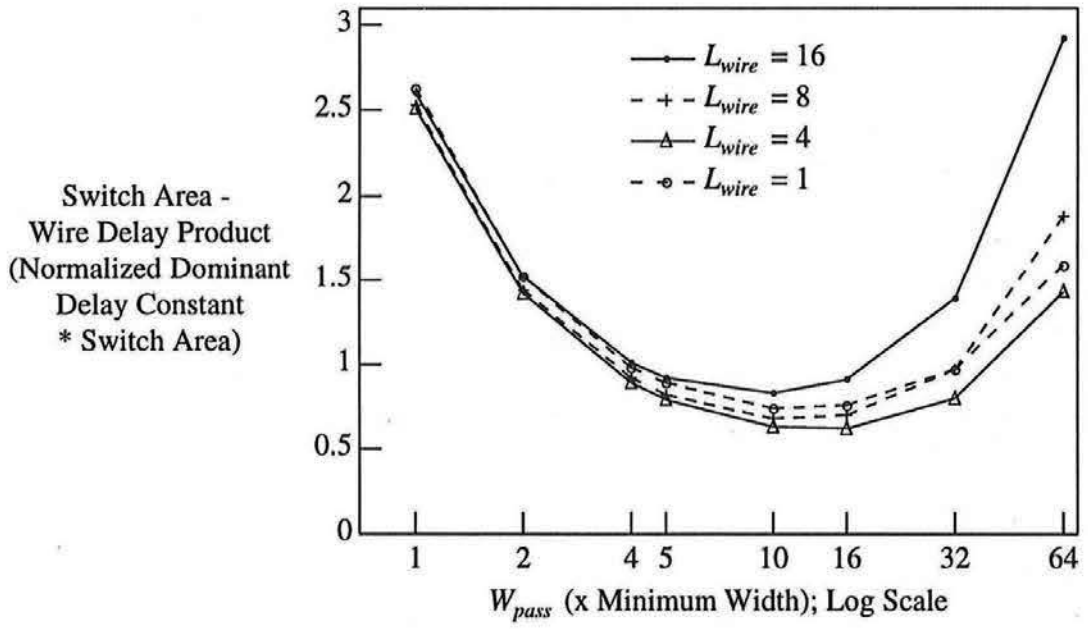


FIGURE C.4 Switch area - wire delay product vs. routing pass transistor width.

of an FPGA by minimizing the routing switch area - wire delay product for each routing resource, and this is the approach we take to routing transistor sizing.

Figure C.4 plots the area of a routing switch multiplied by the dominant wire delay constant as a function of pass transistor width. Again, curves for several different routing wire lengths are shown, and the switch area - wire delay product of each is normalized by dividing by the wire length so they can all be plotted on the same axes. For wire lengths of 1, 4, or 8 logic blocks, transistor widths of 10 and 16 times the minimum are essentially tied for the best area - delay product. For a wire of length 16, a pass transistor width of 10 is clearly preferable to a width of 16. Consequently we use ten times minimum width (i.e. 7 μm width) pass transistor routing switches throughout Chapter 7.

Although we assumed minimum spacing metal 3 wiring in this section, laying out routing wires in metal 2 or using a different metal spacing does not significantly change the point at which the best switch area - wire delay product occurs.

C.2 Sizing Tri-State Buffer Routing Switches

To determine the best size of tri-state buffers in the routing, we followed a procedure identical to that of the prior section. The wire simulated is the same as that in Figure C.1, except every pass transistor between two routing wires is replaced by two tri-state buffers — one in each direction. We assume that one buffer chain is shared by the three tri-state buffers at each of the two end points of a wire; i.e. we are assuming the fan-out-buffer-sharing case described in Appendix B.1. Once again the routing wire is fabricated in metal 3 using the minimum width and spacing.

Figure C.5 shows the wire delay divided by the wire length (i.e. the delay to pass one logic block) versus the size of the tri-state routing buffers. A buffer of minimum size uses minimum width nMOS transistors, while the pMOS pull-up is 1.9 times minimum width, as described in Appendix B.1. Also recall from Appendix B.1 that the larger buffers are multi-stage buffers, and the stage ratio is kept as close to 4 as possible. As buffer size increases from the minimum size, speeds improve for all wire lengths. Once the buffer is larger than four times the minimum, however, the speed of length 1 wires starts to degrade. This occurs because the increase in buffer intrinsic delay as the buffer grows is larger than the decrease in the time it takes the larger buffer to discharge the routing wire (and attached switch) capacitance. Longer wires continue to see some speed improvement until the buffer size reaches 16 times the minimum (for length 4 wires) or 32 times the minimum (for length 8 and 16 wires).

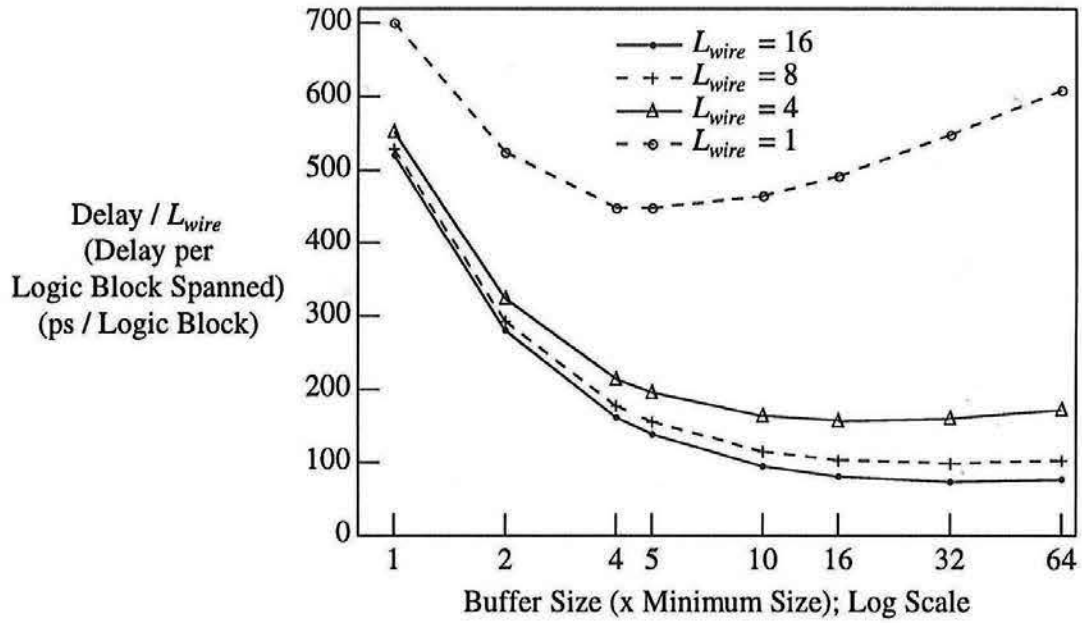


FIGURE C.5 Delay per logic block spanned for buffered routing wires vs. buffer size.

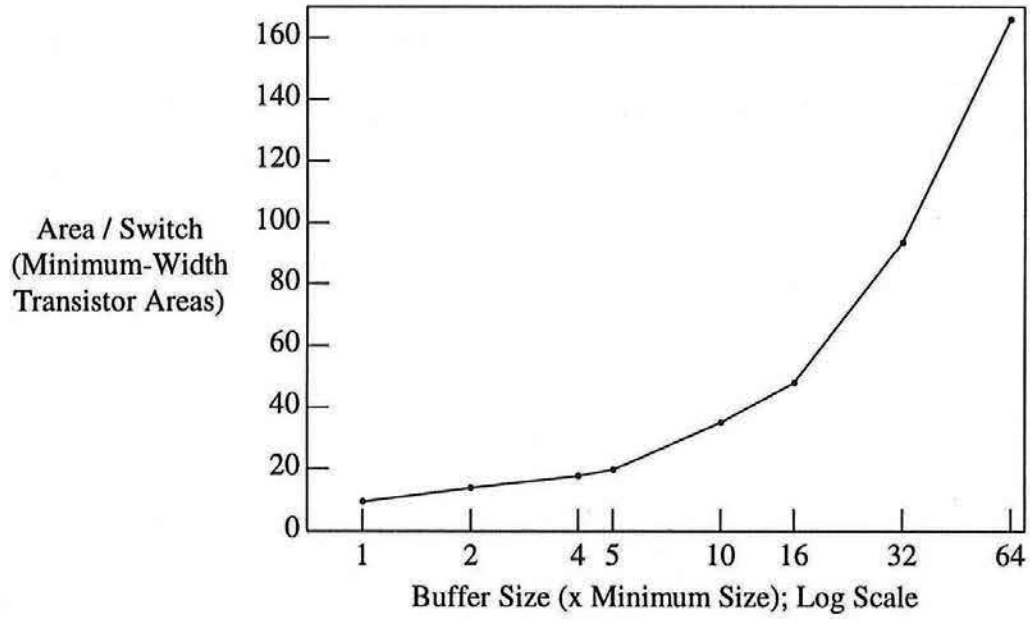


FIGURE C.6 Area of a tri-state buffer switch (including controlling SRAM bit) vs. drive strength.

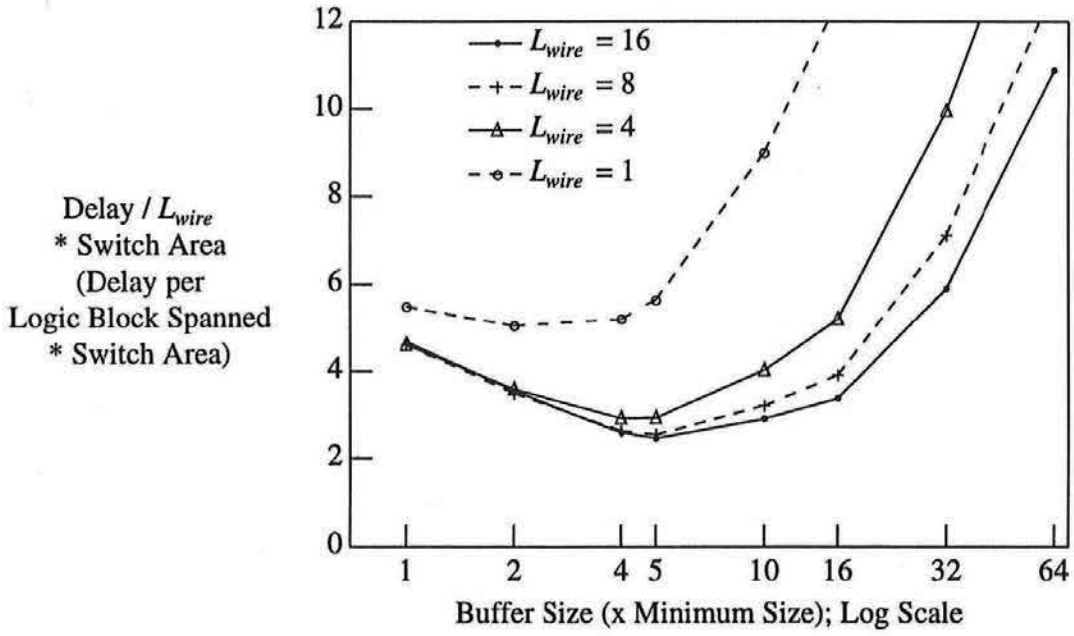


FIGURE C.7 Switch area - wire delay product versus routing tri-state buffer size.

Figure C.6 shows how the area of each tri-state buffer routing switch (including the controlling SRAM bit) varies with the size of the tri-state buffer. Once again, we want to find the buffer size closest to the "knees" of both the area and the delay curves, and once again we take this best trade-off point to be the minimum of the wire delay - routing switch area product. Figure C.7 shows the wire delay - routing switch area product curve for four different routing wire lengths. For wires of length 4, 8 and 16 the best switch area - wire delay product occurs with a buffer that is five times the minimum size. For a wire of length 1, the best area-delay product occurs with a buffer size of only twice the minimum. Since the area-delay product for a wire of length 1 is not much worse with a buffer size of five times the minimum, however, we make all tri-state routing buffers five times the minimum size throughout Chapter 7.

It is interesting to note that the best routing pass transistor width was ten times the minimum, while the best tri-state buffer size is only five times the minimum. There are two reasons for this behaviour. First, as the size of a tri-state buffer is increased, more stages are added to the buffer chain. Thus some of the speed gained by the increased buffer drive strength is counteracted by the increased intrinsic delay of the buffer. Second, since a tri-state buffer contains several transistors, it consumes more area at a given size than a pass transistor. Consequently, as a tri-state buffer is sized up, it more rapidly swamps the fixed area overhead due to its controlling SRAM bit, so its area growth is closer to linear in the buffer size than that of a pass transistor.

C.3 Tri-State Buffers in Output Pin Connection Blocks

The experiments in the prior two sections determined the proper transistor sizes for routing switches contained in FPGA switch blocks. We must also determine a proper size for the tri-state buffers connecting logic block output pins to routing tracks. It makes little sense to connect a routing wire to a logic block output pin with a large tri-state buffer if the routing switches the wire uses to connect to other routing wires are small. The large tri-state buffer from the output pin will make the connection from the logic block to the routing wire fast, but subsequent connections through switch blocks will be slow. Consequently, the extra area used to create a large tri-state buffer from the output pin to the routing wire is largely wasted.

On the other hand, it also makes little sense to connect a logic block output pin to a routing wire with a tri-state buffer that is smaller than the switch block routing switches on this wire. The connection from the output pin to the routing wire will be slow, degrading the speed of what was intended to be a fast routing resource. Consequently, in Chapters 6 and 7 we always make the tri-state buffer connecting a logic block output pin to a routing wire the same size as the routing switches the wire uses to connect to other routing wires. This means the tri-state buffer connecting output pins to pass-transistor-switched wires are ten times the minimum size, while those connecting to buffer-switched wires are only five times the minimum size.

C.4 Metal Width and Spacing

The results in Appendices C.1 and C.2 assumed that routing wires used the minimum metal width and spacing. Increasing the spacing between metal wires reduces the metal capacitance, while increasing the metal width reduces the metal resistance, at the cost of some increase in the metal capacitance. Of course, increasing either the metal width or the metal spacing increases the metal pitch; this may cause in an increase in the FPGA area (if the metal area becomes too large to fit over the transistor area). In this section we determine if FPGA routing wires benefit from greater than minimum metal width or spacing. We again assume that routing wires are laid out in metal 3.

Figure C.8 shows the switch area - dominant delay constant product for a wire of length 16 connected by pass transistor routing switches as a function of the pass transistor width. The three curves in Figure C.8 show the area-delay product achievable under three different metal widths and spacings. The solid curve occurs with minimum metal width and spacing; the dashed curve occurs with minimum width and

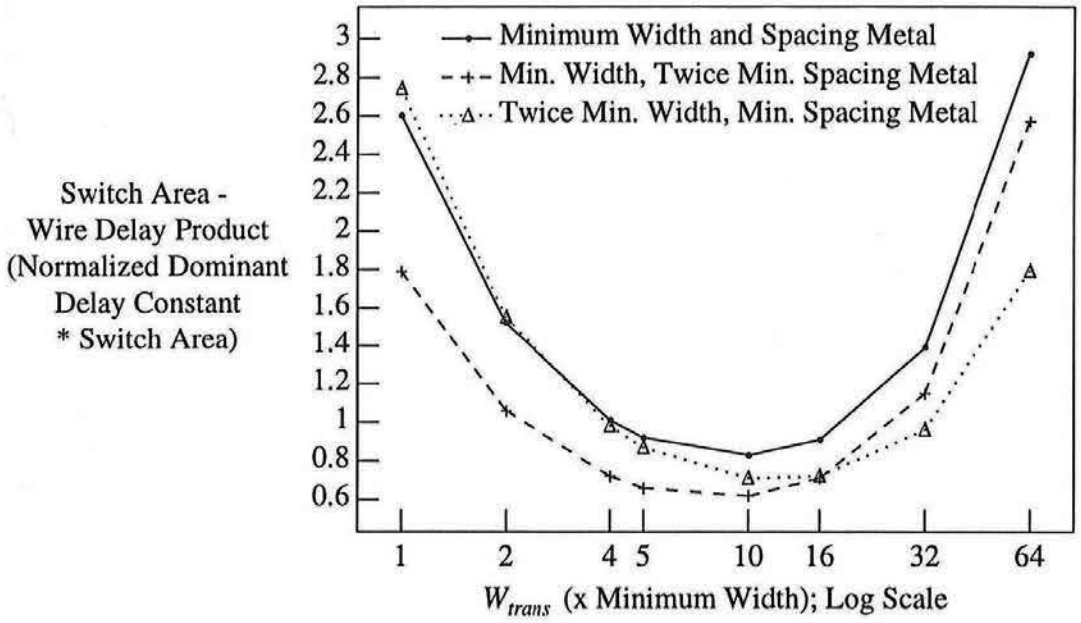


FIGURE C.8 Effect of metal width and spacing on the switch area - wire delay product for a length 16 pass-transistor-switched wire.

twice the minimum spacing (leading to a metal pitch of 1.5 times the minimum metal pitch); and the dotted curve occurs with twice the minimum width and minimum spacing (also leading to a metal pitch of 1.5 times the minimum metal pitch). Notice first that the minimum area-delay product occurs at a pass transistor width of ten times the minimum width for all three curves, showing that the transistor sizing result we obtained in Appendix C.1 is not very sensitive to the metal width and spacing assumed. Secondly, notice that the twice-minimum-spacing curve has an area-delay product at this minimum point that is 13% better than the area-delay product of the twice-minimum-width curve. The entire 13% area-delay product improvement is due to lower delay, since both curves are using the same size of routing switch (ten times minimum width) at this point. Clearly, spacing out metal wires yields greater speed improvements than widening wires in the routing transistor width regime of interest, and is hence a better use of metal area. Note also that we have intentionally presented results for the case that is most favourable to widening the routing wires — a long (length 16) wire. Widening the routing wires performs even more poorly, relative to increasing the wire spacing, for shorter wires.

Figure C.9 shows how the switch area - wire delay product varies with metal width and spacing for a length 16 wire that is driven by tri-state buffer routing switches. Once again, the solid line uses the minimum metal width and spacing, while the dashed line uses twice the minimum metal spacing, and the dotted line uses twice the minimum metal width. Notice that the minimum switch area - wire delay product

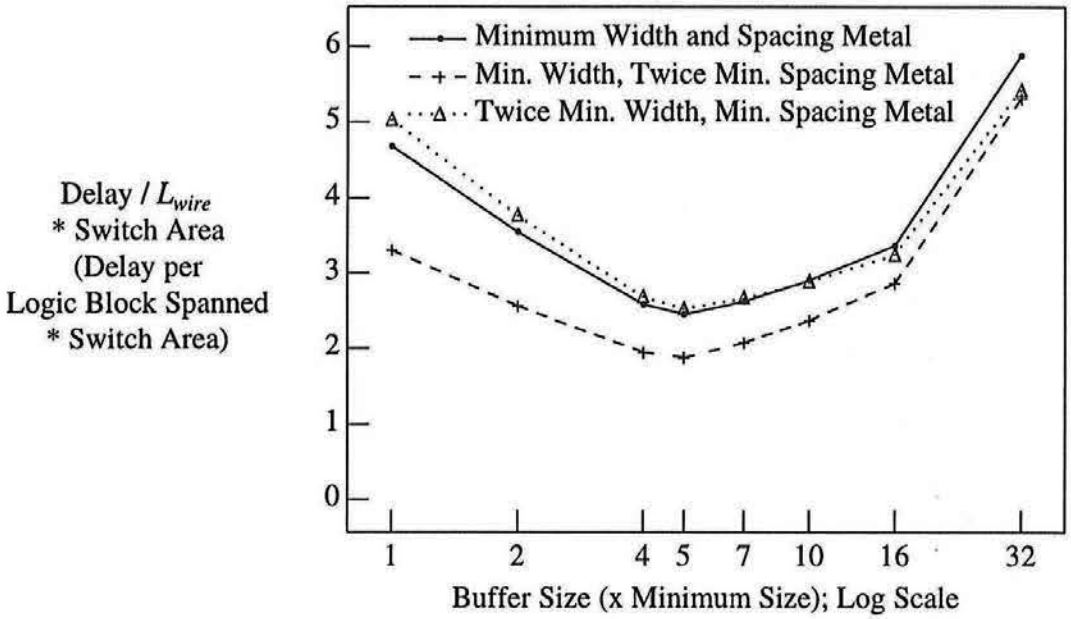


FIGURE C.9 Effect of metal width and spacing on the switch area - wire delay product for a a length 16 buffered routing wire.

occurs at a buffer size of five times the minimum for each of the three curves, indicating that our buffer sizing results of Appendix C.2 are also fairly insensitive to the metal width and spacing of the routing wires. The routing wire using twice the minimum spacing now has a 26% better area-delay product at the minimum area-delay point than the routing wire that uses twice the minimum width. Once again the entire 26% improvement in the area-delay product is due to delay improvement, since both the curves are using the same buffer size (five times minimum size) at this point. Buffered routing switches prevent the "build-up" of wiring resistance when several routing wires and switches are connected in series to form a longer connection. Since the sole benefit of widening a routing wire is a reduction in the wire metal resistance, the fact that widening a routing wire is an even less effective speed optimization with buffered routing wires than with pass-transistor-switched wires is understandable. In fact, widening the routing wire has slightly increased the area-delay product at the minimum point versus using a minimum width wire. Notice that we have again presented results for a long (length 16) routing wire — shorter wires favour increasing the wire spacing over increasing the wire width even more.

It is understandable that increasing the metal spacing improves the routing wire speed, since a wider metal spacing reduces the metal capacitance. On the other hand, it may seem surprising that increasing the metal width is so ineffective, and in fact degrades speed for buffered wires or short pass-transistor-switched wires. Recall that while increasing metal width decreases the metal resistance, it increases the metal capacitance. In standard cell designs this increase in metal capacitance is not an over-

riding issue — one can simply increase the size of the buffer driving the wire to compensate. In FPGA routing, however, we have not one buffer or pass transistor driving a wire, but many possible drivers hanging off the wire along its length. The area cost of sizing up all these buffers or pass transistors to compensate for the increased metal capacitance is significant, and cannot be ignored. Normally these buffers or pass transistors are small enough that their equivalent resistance is considerably higher than the metal resistance of even a fairly long, minimum width wire. Consequently, when one is unwilling to tolerate a large increase in the routing area, increasing the spacing between metal wires is a more effective method of speeding up an FPGA's routing than increasing the wire width (in a 0.35 μm process — future process shrinks may eventually make wider wires more attractive). Accordingly, we investigate the effect of increasing the spacing between routing wires on FPGA speed (in Section 7.6), but we use minimum width metal wires throughout this work.

References

[1] S. Brown, R. Francis, J. Rose and Z. Vranesic, *Field-Programmable Gate Arrays*, Kluwer Academic Publishers, 1992.

[2] V. Betz and J. Rose, "Directional Bias and Non-Uniformity in FPGA Global Routing Architectures," *ICCAD*, 1996, pp. 652 - 659.

[3] V. Betz and J. Rose, "Effect of the Prefabricated Routing Track Distribution on FPGA Area-Efficiency," *IEEE Trans. on VLSI*, Sept. 1998, pp. 445 - 456.

[4] Xilinx Inc., *The Programmable Logic Data Book*, 1994.

[5] Lucent Technologies, *FPGA Data Book*, 1998.

[6] Actel Inc., *FPGA Data Book and Design Guide*, 1994.

[7] Altera Inc., *Data Book*, 1998.

[8] V. Betz and J. Rose, "Cluster-Based Logic Blocks for FPGAs: Area-Efficiency vs. Input Sharing and Size," *CICC*, 1997, pp. 551 - 554.

[9] V. Betz and J. Rose, "How Much Logic Should Go in an FPGA Logic Block?," *IEEE Design and Test Magazine*, Spring 1998, pp. 10 - 15.

[10] A. Marquardt, V. Betz and J. Rose, "Using Cluster-Based Logic Blocks to Improve FPGA Speed and Density," *To appear in 1999 ACM Symp. on FPGAs*.

[11] D. Tavana, W. Lee, S. Young, and B. Fawcett, "Logic Block and Routing Considerations for a New SRAM-Based FPGA Architecture," *CICC*, 1995, ppp. 24.6.1 - 24.6.4.

[12] D. Bursky, "Programmable Arrays Mix FPGA and ASIC Blocks," *Electronic Design*, Oct. 14, 1996, pp. 69 - 74.

[13] Vantis Corporation, "VF1 Field Programmable Gate Arrays," *Preliminary Data Sheet*, 1998.

[14] V. Betz and J. Rose, "FPGA Routing Architecture: Segmentation and Buffering to Optimize Speed and Density," *To appear in 1999 ACM Int. Symp. on FPGAs*.

[15] S. Trimberger, "A Reprogrammable Gate Array and Applications," *Proceedings of the IEEE*, July 1993, pp. 1030 - 1041.

[16] J. Rose and D. Hill, "Architectural and Physical Design Challenges for One-Million Gate FPGAs and Beyond," *ACM Int. Symp. on FPGAs*, 1997, pp. 129 - 132.

[17] V. Betz and J. Rose, "VPR: A New Packing, Placement and Routing Tool for FPGA Research," *Int. Workshop on Field-Programmable Logic and Applications*, 1997, pp. 213 - 222.

[18] V. Betz, "Architecture and CAD for Speed and Area Optimization of FPGAs," *Ph.D. Thesis*, University of Toronto, 1998.

[19] A. Marquardt, *M.A.Sc. Thesis*, Univeristy of Toronto, *in Preparation.*

[20] J. Rose, A. El Gamal and A. Sangiovanni-Vincentelli, "Architecture of Field-Programmable Gate Arrays," *Proceedings of the IEEE*, July 1993, pp. 1013 - 1029.

[21] P. Chow, S. Seo, J. Rose, K. Chung, G. Paez and I. Rahardja, "The Design of an SRAM-Based Field-Programmable Gate Array, Part II: Circuit Design and Layout," *To appear in IEEE Trans. on VLSI.*

[22] M. Khellah, S. Brown and Z. Vranesic, "Modelling Routing Delays in SRAM-based FPGAs," *Proc. Canadian Conf. on VLSI*, 1993, pp. 6B.13 - 6B.18.

[23] J. Greene, E. Hamdy and S. Beal, "Antifuse Field Programmable Gate Arrays," *Proceedings of the IEEE*, July 1993, pp. 1042 - 1056.

[24] S. Brown, "An Overview of Technology, Architecture and CAD Tools for Programmable Logic Devices," *CICC*, 1994, pp. 69 - 76.

[25] J. Rose, R. Francis, D. Lewis and P. Chow, "Architecture of Field-Programmable Gate Arrays: The Effect of Logic Block Functionality on Area Efficiency," *JSSC*, Oct. 1990, pp. 1217 - 1225.

[26] J. He and J. Rose, "Advantages of Heterogenous Logic Block Architectures for FPGAs," *CICC*, 1993, pp. 7.4.1 - 7.4.5.

[27] K. Chung, S. Singh, J. Rose and P. Chow, "Using Hierarchical Logic Blocks to Improve the Speed of Field-Programmable Gate Arrays," *Int. Workshop on Field Programmable Logic and Applications*, 1991.

[28] J. Cong and Y. Hwang, "Boolean Matching for Complex PLBs in LUT-based FPGAs with Application to Architecture Evaluation," *ACM Symp. on FPGAs*, 1998, pp. 27 - 34.

[29] A. Aggarwal and D. Lewis, "Routing Architectures for Hierarchical Field Programmable Gate Arrays," *ICCD*, 1994, pp. 475 - 478.

[30] J. Rose and S. Brown, "Flexibility of Interconnection Structures for Field-Programmable Gate Arrays," *JSSC*, March 1991, pp. 277 - 282.

[31] B. Tseng, J. Rose and S. Brown, "Using Architectural and CAD Interactions to Improve FPGA Routing Architectures," *ACM Workshop on FPGAs*, 1992, pp. 3 - 8.

[32] Y. Chang, D. F. Wong, and C. K. Wong, "Universal Switch Modules for FPGA Design," *ACM Trans. on Design Automation of Electronic Systems*, Jan. 1996, pp. 80 - 101

[33] S. Wilton, "Architectures and Algorithms for Field-Programmable Gate Arrays with Embedded Memories," *Ph.D. Dissertation*, University of Toronto, 1997. (*Available for download from http://www.ee.ubc.ca/~stevew/publications.html*).

[34] J. Greene, V. Roychowdhury, S. Kaptanoglu and A. El Gamal, "Segmented Channel Routing," *DAC*, 1990, pp. 567 - 572.

[35] K. Roy and M. Mehendale, "Optimization of Channel Segmentation for Channelled Architecture FPGAs," *CICC*, 1992, pp. 4.4.1 - 4.4.4.

[36] S. Brown, M. Khellah and G. Lemieux, "Segmented Routing for Speed-Performance and Routability in Field-Programmable Gate Arrays," *Journal of VLSI Design*, Vol. 4, No. 4, 1996, pp. 275 - 291.

[37] M. Khellah, S. Brown and Z. Vranesic, "Minimizing Interconnection Delays in Array-Based FPGAs," *CICC*, 1994, pp. 181 - 184.

[38] S. Brown, M. Khellah and Z. Vranesic, "Minimizing FPGA Interconnect Delays," *IEEE Design and Test Magazine*, Winter 1996, pp. 16 - 23.

[39] P. Chow, S. Seo, J. Rose, K. Chung, G. Paez and I. Rahardja, "The Design of an SRAM-Based Field-Programmable Gate Array, Part I: Architecture," *To appear in IEEE Trans. on VLSI.*

[40] Y. Sun, T. Wang, C. Wong and C. Liu, "Routing for Symmetric FPGA's and FPIC's," *IEEE Trans. on CAD*, Jan. 1997, pp. 20 - 31.

[41] A. Sangiovanni-Vincentelli, A. El Gamal, and J. Rose, "Synthesis Methods for Field-Programmable Gate Arrays," *Proceedings of the IEEE*, July 1993, pp. 1057 - 1083.

[42] R. Brayton, G. Hachtel, and A. Sangiovanni-Vincentelli, "Multilevel Logic Syntesis," *Proceedings of the IEEE*, Feb. 1990, pp. 264 - 300.

[43] R. Francis, J. Rose and Z. Vranesic, "Chortle-crf: Fast Technology Mapping for Lookup Table-Based FPGAs," *DAC*, 1991, pp. 227 - 233.

[44] R. Francis, J. Rose and Z. Vranesic, "Technology Mapping Lookup Table-Based FPGAs for Performance" *ICCAD*, 1991, pp. 568 - 571.

[45] K. C. Chen, J. Cong, Y. Ding, A. Kahng, and P. Trajmar, "DAG-Map: Graph-Based FPGA Technology Mapping for Delay Optimization," *IEEE Design and Test Magzine*, Sept. 1992, pp. 7 - 20.

[46] J. Cong and Y. Ding, "FlowMap: An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-Table Based FPGA Designs," *IEEE Trans. on CAD*, Jan. 1994, pp. 1 - 12.

[47] J. Cong and Y. Ding, "On Area / Depth Trade-Off in LUT-Based FPGA Technology Mapping," *IEEE Trans. on VLSI*, June 1994, pp. 137 - 148.

[48] C. Alpert and A. Kahng, "Recent Directions in Netlist Partitioning: A Survey," *Integration, the VLSI Journal*, Vol. 19 (1-2), 1995, pp. 1 - 81.

[49] J. Cong, H. Li, S. Lim, T. Shibuya, and D. Xu, "Large Scale Circuit Partitioning with Loose / Stable Net Removal and Signal Flow Based Clustering," *ICCAD*, 1997, pp. 441 - 446.

[50] L. Hagen and A. Kahng, "A New Approach to Effective Circuit Clustering," *ICCAD*, 1992, pp. 422 - 427.

[51] J. Cong and M. Smith, "A Parallel Bottom-up Clustering Algorithm with Applications to Circuit Partitioning in VLSI Design," *DAC*, 1993, pp. 755 - 760.

[52] G. Saucier, D. Brasen, J. Hiol, "Partitioning with Cone Structures," *ICCAD* 1993, pp. 236 - 239.

[53] L. Hagen and A. Kahng, "Fast Spectral Methods for Ratio Cut Partitioning and Clustering," *ICCAD*, 1991, pp. 10 - 13.

[54] C. Alpert and A. Kahng, "Geometric Embeddings for Faster and Better Multi-Way Netlist Partitioning," *DAC*, 1993, pp. 743 - 748.

[55] C. Alpert and S. Z. Yao, "Spectral Partitioning: The More Eigenvectors, The Better," *DAC*, 1995, pp. 195 - 200.

[56] C. Alpert and A. Kahng, "A General Framework for Vertex Ordering, With Applications to Netlist Clustering," *ICCAD*, 1994, pp. 63 - 67.

[57] E. Lawler, K. Levitt and J. Turner, "Module Clustering to Minimize Delay in Digital Networks," *IEEE Trans. on Computers*, Jan. 1966, pp. 47 - 57.

[58] R. Murgai, R. Brayton, and A. Sangiovanni-Vincentelli, "On Clustering for Minimum Delay / Area," *ICCAD*, 1991, pp. 6 - 9.

[59] R. Rajaraman and D. F. Wong, "Optimal Clustering for Delay Minimization," *DAC*, 1993, pp. 309 - 314.

[60] H. Yang and D. Wong, "Area/Pin-Constrained Circuit Clustering for Delay Minimization," *ACM Int. Workshop on FPGAs*, 1994, pp. 7.1.1 - 7.1.10.

[61] J. Cong, J. Peck, and Y. Ding, "RASP: A General Logic Synthesis System for SRAM-based FPGAs," *ACM Symp. on FPGAs*, 1996, pp. 137 - 143.

[62] D. West, *Introduction to Graph Theory,* Prentice Hall, 1996, p. 130.

[63] H. Gabow and R. Tarjan, "Faster scaling algorithms for general graph-matching problems," *Journal of the ACM*, Oct. 1991, pp. 815-853.

[64] H. Shin and C. Kim, "A Simple Yet Effective Technique for Partitioning," *IEEE Trans. on VLSI*, Sept. 1993, pp. 380 - 386.

[65] A. Dunlop and B. Kernighan, "A Procedure for Placement of Standard-Cell VLSI Circuits," *IEEE Trans. on CAD*, Jan. 1985, pp. 92 - 98.

[66] D. Huang and A. Kahng, "Partitioning-Based Standard-Cell Global Placement with an Exact Objective," *ACM Symp. on Physical Design*, 1997, pp. 18 - 25.

[67] J. Rose, W. Snelgrove and Z. Vranesic, "ALTOR: An Automatic Standard Cell Layout Program," *Canadian Conf. on VLSI*, 1985, pp. 169 - 173.

[68] J. Kleinhans, G. Sigl, F. Johannes and K. Antreich, "Gordian: VLSI Placement by Quadratic Programming and Slicing Optimization," *IEEE Trans. on CAD*, March 1991, pp. 356 - 365.

[69] G. Sigl, K. Doll and F. Johannes, "Analytical Placement: A Linear or a Quadratic Objective Function?," *DAC*, 1991, pp. 427 - 432.

[70] C. Alpert, T. Chan, D. Huang, A. Kahng, I. Markov, P. Mulet and K. Yan, "Faster Minimization of Linear Wirelength for Global Placement," *ACM Symp. on Physical Design*, 1997, pp. 4 - 11.

[71] C. Alpert, T. Chan, D. Huang, I. Markov and K. Yan, "Quadratic Placement Revisited," *DAC*, 1997, pp. 752 - 757.

[72] A. Srinivasan, "An Algorithm for Performance-Driven Initial Placement of Small-Cell ICs," *DAC*, 1991, pp. 636 - 639.

[73] A. Srinivasan, K. Chaudhary and E. Kuh, "Ritual: A Performance Driven Placement Algorithm for Small Cell ICs," *ICCAD*, 1991, pp. 48 - 51.

[74] B. Riess and G. Ettelt, "Speed: Fast and Efficient Timing Driven Placement," *IEEE Int. Symp. on Circuits and Systems*, 1995, pp. 377 - 380.

[75] K. Doll, F. Johannes and G. Sigl, "Domino: Deterministic Placement Improvement with Hill-climbing Capabilities," *Proc. VLSI*, 1991, pp. 3b.1.1 - 3b.1.10.

[76] S. Kirkpatrick, C. Gelatt and M. Vecchi, "Optimization by Simulated Annealing," *Science*, May 13, 1983, pp. 671 - 680.

[77] C. Sechen and A. Sangiovanni-Vincentelli, "The TimberWolf Placement and Routing Package," *JSSC*, April 1985, pp. 510 - 522.

[78] C. Sechen and A. Sangiovanni-Vincentelli, "TimberWolf3.2: A New Standard Cell Placement and Global Routing Package," *DAC*, 1986, pp. 432 - 439.

[79] C. Sechen and K. Lee, "An Improved Simulated Annealing Algorithm for Row-Based Placement," *ICCAD*, 1987, pp. 478 - 481.

[80] W. Sun and C. Sechen, "Efficient and Effective Placement for Very Large Circuits," *IEEE Trans. on CAD*, March 1995, pp. 349 - 359.

[81] W. Swartz and C. Sechen, "Timing Driven Placement for Large Standard Cell Circuits," *DAC*, 1995, pp. 211 - 215.

[82] M. Huang, F. Romeo, and A. Sangiovanni-Vincentelli, "An Efficient General Cooling Schedule for Simulated Annealing," *ICCAD*, 1986, pp. 381 - 384.

[83] J. Lam and J. Delosme, "Performance of a New Annealing Schedule," *DAC*, 1988, pp. 306 - 311.

[84] W. Swartz and C. Sechen, "New Algorithms for the Placement and Routing of Macro Cells," *ICCAD*, 1990, pp. 336 - 339.

[85] C. Ebeling, L. McMurchie, S. A. Hauck and S. Burns, "Placement and Routing Tools for the Triptych FPGA," *IEEE Trans. on VLSI*, Dec. 1995, pp. 473 - 482.

[86] S. Nag and R. Rutenbar, "Performance-Driven Simultaneous Place and Route for Island-Style FPGAs," *ICCAD*, 1995, pp. 332 - 338.

[87] S. Nag and R. Rutenbar, "Performance-Driven Simultaneous Place and Route for Row-Based FPGAs," *DAC*, 1994, pp. 301 - 307.

[88] M. Alexander, J. Cohoon, J. Ganley and G. Robins, "Performance-Oriented Placement and Routing for Field-Programmable Gate Arrays," *European Design Automation Conf.*, 1995, pp. 80 - 85.

[89] C. Cheng, "RISA: Accurate and Efficient Placement Routability Modeling," *ICCAD*, 1994, pp. 690 - 695.

[90] M. Alexander, J. Cohoon, J. Ganley and G. Robins, "An Architecture-Independent Approach to FPGA Routing Based on Multi-Weighted Graphs," *European Design Automation Conf.*, 1994, pp. 259 - 264.

[91] Y.-L. Wu, M. Marek-Sadowska, "An Efficient Router for 2-D Field-Programmable Gate Arrays," *European Design Automation Conf.*, 1994, pp. 412 - 416.

[92] Y.-L. Wu, M. Marek-Sadowska, "Orthogonal Greedy Coupling -- A New Optimization Approach to 2-D FPGA Routing," *DAC*, 1995, pp. 568 - 573.

[93] M. J. Alexander, G. Robins, "New Performance-Driven FPGA Routing Algorithms," *DAC*, 1995, pp. 562 - 567.

[94] Y.-S. Lee, A. Wu, "A Performance and Routability Driven Router for FPGAs Considering Path Delays," *DAC*, 1995, pp. 557 - 561.

[95] J. Frankle, "Iterative and Adaptive Slack Allocation for Performance-Driven Layout and FPGA Routing," *DAC*, 1992, pp. 536 - 542.

[96] M. Placzewski, "Plane Parallel A* Maze Router and Its Application to FPGAs," *DAC*, 1992, pp. 691 - 697.

[97] J. S. Rose, "Parallel Global Routing for Standard Cells," *IEEE Trans. on CAD*, Oct. 1990, pp. 1085 - 1095.

[98] Y. Chang, S. Thakur, K. Zhu and D. Wong, "A New Global Routing Algorithm for FPGAs," *ICCAD*, 1994, pp. 356 - 361.

[99] S. Brown, J. Rose, Z. G. Vranesic, "A Detailed Router for Field-Programmable Gate Arrays," *IEEE Trans. on CAD*, May 1992, pp. 620 - 628.

[100] G. Lemieux, S. Brown, "A Detailed Router for Allocating Wire Segments in FPGAs," *ACM/SIGDA Physical Design Workshop*, 1993, pp. 215 - 226.

[101] G. Lemieux, S. Brown, D. Vranesic, "On Two-Step Routing for FPGAs," *ACM Symp. on Physical Design*, 1997, pp. 60 - 66.

[102] C. Y. Lee, "An Algorithm for Path Connections and its Applications," *IRE Trans. Electron. Comput.*, Vol. EC=10, 1961, pp. 346 - 365.

[103] E. Dijkstra, "A Note on Two Problems in Connexion with Graphs," *Numer. Math.*, Vol. 1, 1959, pp. 269 - 271.

[104] R. Nair, "A Simple Yet Effective Technique for Global Wiring," *IEEE Trans. on CAD*, March 1987, pp. 165-172.

[105] F. Rubin, "The Lee Path Connection Algorithm," *IEEE Trans. Computers*, Sept. 1974, pp. 907 - 914.

[106] S. Trimberger, *Field-Programmable Gate Array Technology*, Kluwer Academic Publishers, 1994.

[107] J. Rubinstein, P. Penfield and M. Horowitz, "Signal Delay in RC Tree Networks," *IEEE Trans. on CAD*, 1983, pp. 202 - 211.

[108] J. Cong, L. He, C. Koh and P. Madden, "Performance Optimization of VLSI Interconnect Layout," *Integration, The VLSI Journal*, Vol. 21, 1996, pp. 1 - 94.

[109] S. Prasitjutrakul and W. Kubitz, "A Timing-Driven Global Router for Custom Chip Design," *ICCAD*, 1990, pp. 48 - 51.

[110] C. Alpert, T. Hu, J. Huang and A. Kahng, "A Direct Combination of the Prim and Dijkstra Constructions for Improved Performance-Driven Global Routing," *IEEE Int. Symp. on Circuits and Systems*, 1993, pp. 1869 - 1872.

[111] K. Boese, A. Kahng and G. Robins, "High-Performance Routing Trees with Identified Critical Sinks," *DAC*, 1993, pp. 182 - 187.

[112] T. Okamoto and J. Cong, "Buffered Steiner Tree Construction with Wire Sizing for Interconnect Layout Optimization," *ICCAD*, 1996, pp. 44 - 49.

[113] J. Cong, A. Kahng, G. Robins, M. Sarrafzadeh and C. Wong, "Provably Good Performance-Driven Global Routing," *IEEE Trans. on CAD*, June 1992, pp. 739 - 752.

[114] M. Kang, W. Dai, T. Dillinger and D. LaPotin, "Delay Bounded Buffered Tree Construction for Timing Driven Floorplanning," *ICCAD*, 1997, pp. 707 - 712.

[115] J. Lillis, T. Lin and C. Ho, "New Performance Driven Routing Techniques with Explicit Area / Delay Tradeoff and Simultaneous Wire Sizing," *DAC*, 1996, pp. 395 - 400.

[116] J. Cong, K. Leung and D. Zhou, "Performance-Driven Interconnect Design Based on Distributed RC Delay Model," *DAC*, 1993, pp. 606 - 611.

[117] A. Lim, S. Cheng and C. Wu, "Performance Oriented Rectilinear Steiner Trees," *DAC*, 1993, pp. 171 - 176.

[118] X. Hong, T. Xue, E. Kuh, C. Cheng and J. Huang, "Performance-Driven Steiner Tree Algorithms for Global Routing," *DAC*, 1993, pp. 177 - 181.

[119] K. Boese, A. Kahng, B. McCoy and G. Robins, "Rectilinear Steiner Trees with Minimum Elmore Delay," *DAC*, 1994, pp. 381 - 386.

[120] A. Vittal and M. Marek-Sadowska, "Minimal Delay Interconnect Design Using Alphabetic Trees," *DAC*, 1994, pp. 392 - 396.

[121] J. Cong and C. Koh, "Interconnect Layout Optimization Under Higher-Order RLC Model," *ICCAD*, 1997, pp. 713 - 720.

[122] L. Kannan, P. Suaris and H. Fang, "A Methodology and Algorithms for Post-Placement Delay Optimization," *DAC*, 1994.

[123] J. Lillis, C. Cheng and T. Lin, "Optimal Wire Sizing and Buffer Insertion for Low Power and a Generalized Delay Model," *ICCAD*, 1995, pp. 138 - 143.

[124] K. Sato and M. Kawarabayashi, "Post-Layout Optimization for Deep Submicron Design," *DAC*, 1996, 740 - 745.

[125] C. Alpert and A. Devgan, "Wire Segmenting for Improved Buffer Insertion," *DAC*, 1997, pp. 588 - 593.

[126] C. Chu and D. Wong, "A New Approach to Simultaneous Buffer Insertion and Wire Sizing," *ICCAD*, 1997, pp. 614 - 621.

[127] J. Swartz, V. Betz and J. Rose, "A Fast Routability-Driven Router for FPGAs," *ACM Symp. on FPGAs*, 1998, pp. 140 - 149.

[128] W. Elmore, "The Transient Response of Damped Linear Networks with Particular Regard to Wideband Amplifiers," *Journal of Applied Physics*, Jan. 1948, pp. 55 - 63.

[129] K. Boese, A. Kahng, B. McCoy and G. Robins, "Fidelity and Near-Optimality of Elmore-Based Routing Constructions," *ICCD*, 1993, pp. 81 - 84.

[130] J. Cong and L. He, "Optimal Wiresizing for Interconnects with Multiple Sources," *ACM Trans. on Design Automation of Electronic Systems*," Oct. 1996, pp. 478 - 511.

[131] R. Hitchcock, G. Smith and D. Cheng, "Timing Analysis of Computer-Hardware," *IBM Journal of Research and Development*, Jan. 1983, pp. 100 - 105.

[132] P. Hauge, R. Nair and E. Yoffa, "Circuit Placement for Predictable Performance," *ICCAD*, 1987, pp. 88 - 91.

[133] H. Youssef and E. Shragowitz, "Timing Constraints for Correct Performance," *ICCAD*, 1990, pp. 24 - 27.

[134] V. Betz, "VPack and VPR User's Manual," 1998. (*Available for download from http://www.eecg.toronto.edu/~jayar/software/software.html).*

[135] M. Khalid and J. Rose, "The Effect of Fixed I/O Pin Positioning on The Routability and Speed of FPGAs," *Canadian Workshop on Field-Programmable Devices,* 1995, pp. 94 - 102.

[136] S. Yang, "Logic Synthesis and Optimization Benchmarks, Version 3.0," *Tech. Report*, Microelectronics Center of North Carolina, 1991.

[137] H. Hseih, et al, "Third-Generation Architecture Boosts Speed and Density of Field-Programmable Gate Arrays," *CICC*, 1990, pp. 31.2.1 - 31.27.

[138] P. Leventis, *Undergraduate Thesis,* University of Toronto, *In Preparation.*

[139] Xilinx Inc., "XC4000E and XC4000X Series Field-Programmable Gate Arrays," *Data Sheet*, 1997.

[140] J. Swartz, "A High-Speed Timing-Aware Router for FPGAs," *M.A.Sc. Thesis,* University of Toronto, 1998.

[141] B. K. Britton et al., "Second Generation ORCA Architecture Utilizing 0.5 μm Process Enhances the Speed and Usable Gate Capacity of FPGAs," *IEEE Int. ASIC Conf.*, Sept. 1994, pp. 474 - 478.

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

[143] V. Betz and J. Rose, "On Biased and Non-uniform Global Routing Architectures and CAD Tools for FPGAs," Technical Report, University of Toronto, 1996.

[144] M. Feuer, "Connectivity of Random Logic," *IEEE Trans. on Computers*, Jan. 1982, pp. 29 - 33.

[145] C. Matsumoto, "Million-Gate Architecture Will Vie with ASIC-like Approach from Altera, Lucent, and Gatefield," *Electronic Engineering Times*, Oct. 26, 1998, p. 1.

[146] P. Clarke, "Dynachip Claims Speed Breakthrough in its FPGAs," *Electronic Engineering Times*, June 9, 1997, p. 10.

[147] M. Hutton, J. Rose, J. Grossman, and D. Corneil, "Characterization and Parameterized Random Generation of Combinational Benchmark Circuits," *to appear in IEEE Trans. on CAD.*

[148] M. Hutton, J. Rose, D. Corneil, "Generation of Synthetic Sequential Benchmark Circuits," *ACM Symp. on FPGAs*, 1997, pp. 149 - 155.

[149] M. Hutton, J.P. Grossman, J. Rose, D. Corneil, "Characterization and Parameterized Random Generation of Digital Circuits," *DAC*, 1996, pp. 94 - 99.

[150] J. Anderson and S. Brown, "An LPGA with Foldable PLA-Style Logic Blocks," *ACM Symp. on FPGAs*, 1998, pp. 244 - 252.

[151] J. Swartz, V. Betz and J. Rose, "A Routability-Driven and Timing-Aware Router for FPGAs," *Submitted to IEEE Trans. on CAD.*

[152] R. Tessier, "Negotiated A* Routing for FPGAs," *Canadian Workshop on Field-Programmable Devices*, 1998, pp 14 - 19.

[153] Y. Sankar and J. Rose, "Ultra-Fast Placement for FPGAs," *To appear in 1999 ACM Symp. on FPGAs.*

[154] Canadian Microelectronics Corporation, "0.35 μm Mixed-Mode Polycide HSPICE Models," *Confidential Process Documentation*, 1997.

[155] N. Weste and K. Eshraghian, *Principles of CMOS VLSI Design, Second Edition*, Addison-Wesley, 1993.

[156] D. Lewis, *VLSI Arithmetic Structures, Course Notes*, University of Toronto, 1993.

Index

Numerics

10K FPGA 39
2-LUT 13
4000 FPGA 107
4000X FPGA 100
4-LUT 13
5000 FPGA 107
5200 FPGA 37, 141
8K FPGA 39, 141

A

A* 29, 87
Acceptance rate 24
Actel 5, 12, 14, 110, 129
Adaptive annealing schedule 23, 52–55
Aggarwal, A. 14
Alexander, M. 25
Algorithm validation 95–101
Alpert, C. 30
Altera 5, 12, 14, 37, 39, 75, 129, 141, 196
Altor 97
Analytic placement 22
Annealing schedule 23, 52–55
Antifuse 16
Architecture 11
  Description file 65
  Generation 64–75
  Parameterization 64–75
  Parameters 109
  Routing, see Routing Architecture
Area efficiency 13, 109, 142–147
Area metric 17
Area model 2, 132–134, 157, 211
Area-delay product 136, 223
Arrival time 33
Aspect ratio 113
Attraction 41, 43, 44

B

Base cost 76, 90, 91
BaseBLECrit 44
Basic logic elements 38
BendCost 76
Betz, V. 8
BLE 38, 39
Bounding box 23, 56–58, 79
Bounding box incremental updates 58
Brown, S. 14, 15, 17
Buffer 80, 134, 166, 167–175, 210–212
Buffer insertion 87
Buffer model 31
Buffer scaling 138
Buffer sizing 225–228

C

Capacitance 81, 94
Capacity ratio 116
Center/Edge Capacity Ratio 116
Channel capacity 56, 107
Channel segment 26
Channel width 64, 67
Cheng, C. 25
Chow, P. 17
Chung, K. 13
Closeness 21
Cluster 38, 127–148
Cluster inputs 39, 139–141
Cluster occupancy 41
Cluster seed 43
Cluster size 40, 138–148
Cluster-based logic block 4, 37, 127–148
Clustering 20, 37–47
CMOS transistor 219
Compile time 5, 147
Computational complexity
  Bounding box update 58
  Net routing algorithm 90

T-VPack 47
VPack 42
Conclusions 191–198
Cong, J. 14, 21
Congestion 26, 85, 92
Congestion avoidance 77
Congestion memory 27
Connection block 14, 66, 70, 96, 212
Connection block flexibility 15, 141, 165
Connection block internal population, see Routing architecture : Internal population
ConnectionCriticality 43
Connectivity 68
Contributions 191–198
Coordinate system 74
Cost function
Linear congestion 55
Non-linear congestion 56
Coupling capacitance 183
Critical path 27, 34
Criticality 27, 44, 83
Ctotal 81

D
Deep-submicron 1
Delay 26, 67, 85, 145
Delay estimate 31
Delay extraction 94, 217–220
Delay model 2, 31, 80–90, 134–136, 156
Delosme, J. 24, 52
Detailed routing 26
Detailed routing architecture 5, 6, 15, 151–190
Dijkstra, E. 26
Directed search 29, 86
Directional bias 106, 110–115
Disjoint switch block 72, 159
Doglegs 96
Drive strength 133
Dynamic base cost 90
Dynamic costing 92
Dynamic weighting 29

E
Ebeling, C. 24, 29, 78
Elmore delay 30, 31, 80–90, 156, 218, 222
ExitCriterion 23
ExpectedCost 86

F
Fanout 81
Fc, see Connection block flexibility
Flex FPGA 141
FlowMap 98, 107, 130, 155
FPGA architecture 11
FPGA challenge 99
FPGA circuitry 207–220
FPGA programming technology 11
FPR 25, 97
Fraction of moves accepted 53
Fs, see Switch block flexibility
Full-perimeter pin positioning 110, 112
Fully-connected, logic cluster 39, 141
Future work 191–198

G
Gate boosting 208
General-purpose routing 43
Global routing 26, 69, 76
Global routing architecture 3, 15, 55, 105–126
Global-detailed routing 26, 76
Graphic visualization 199–203
Greene, J. 16

H
Hard-wired logic block 13
He, J. 13
hfac 78, 92
Hierarchical 14
Hill-climbing 23, 41, 42
Historical congestion 27, 76
Horizontal constraint 72
Huang, M. 23, 52, 54
Hwang, Y. 14

I
I, see Cluster inputs
I/O pads 51
I/O-channel 122
Incremental bounding box evaluation 58
Incremental net bounding box updates 58
InitialT 23
InnerLoopCriterion 23
InnerNum 52
InputPathsAffected 46
Input-pin Doglegs 96
Inputs required 139–141
Inter-cluster 145
InterClusterConnectionDelay 43

Interconnect delay 6
Internal population 17, 66, 175–182
Intra-cluster 145
IntraClusterConnectionDelay 43
Intrinsic delay 31
Island-style 5, 14, 17, 18, 154

K
K (LUT size) 39
Kim, C. 21
k-input LUT 13

L
Labelling, clustering via 20
Lam, J. 24, 52
Lee, Y. 99
Length of a wiring segment 16
Levelizing 94
Lewis, D. 14
Linear congestion cost function 55
Linear delay 29, 80
Local routing 43
Locked I/Os 51
Logic block packing 19, 37–50
Logic block packing (timing-driven) 43–47
Logic block structure 214
Logic cluster 38, 39, 127–148
Logically-equivalent logic block pins 68, 141
LogicDelay 43
Long line 16
Long routing wires 7
Look-up table 215
Low-stress routing 101, 130
LSI Logic 133
Lucent 5, 12, 14, 106, 120, 129

M
Manhattan distance 147
Marquardt, A. 8
Maze router 26
Mclk 39
Mehendale, M. 16
Metal capacitance 81, 219
Metal resistance 219
Metal sizing 221–231
Metal spacing 228–231
Metal width 228–231
Min-cut placement 22
Minimum-width transistor area 132
MPGA 2, 3
MPGA routing 30
Multi-stage buffer 210–212

N
N, see Cluster size
Nag, S. 25, 29, 90
Nair, R. 27
Netlist 19
nMOS pass transistor 208, 218
Node capacity 69
Non-linear congestion cost function 56
Non-recurring engineering fees 2
Non-uniform routing architecture 115–124
NRE 2

O
Occupancy 78
ORCA 2C FPGA 106
OutputPathsAffected 46

P
Packing 19, 37–50
Parasitic capacitance 17, 31
Partitioning 20
Pass transistor 7, 31, 96, 167–175
Pass transistor chain 81
Pass transistor delay 80
Pass transistor model 31
Pass transistor scaling 138
Pass transistor sizing 221–225
PathCost 29, 86
Pathfinder 26, 76, 80
Pathological switch topology 71
Penfield-Rubinstein 30, 31, 100
pfac 78, 92
Placement 22, 37, 50–60
Placement validation 95–101
pMOS transistor 219
Porosity 25
Present congestion 76
PriorityQueue 86
Probability of acceptance 23, 53
Process modeling 207–220
Programming technology 11

Q
Quality 7, 95–101

R
Rbuf 81

RC-equivalent circuit extraction 217–220
RC-model 31
Rectangular logic block array 113
Required time 34
Re-route 27
Rip-up 27
Rlimit 24, 53
Rmetal 94
Romeo, F. 23
Rose, J. 13, 14, 15
Routability-driven router 26, 29, 76–79
Router 25, 76–93
Routing 25
  Architecture, see Routing architecture
  Channel width 64
  Detailed 26
  Directed search 86
  Directional bias 110–115
  Global 26, 69
  Global-detailed 26, 76
  Iteration 27, 79
  Low-stress 101, 130
  Maze 26
  MPGA 30
  Resource graph 25, 64, 68–70
  Routability-driven 26, 29, 76–79
  Schedule 76, 78, 92–93
  Segment length 138
  Standard cell 30
  Timing-driven 26, 80–93, 101
  Tools 63
  Tree 83
  Validation 95–101
Routing architecture 14, 25, 63–103, 105–126, 137, 151–190
  Buffer 167–175, 210–212
  Buffer sizing 225–228
  Channel capacity 107
  Comparison 185–189
  Detailed 5, 6, 15, 151–190
  Generation 64–75
  Global 3, 15, 55, 105–126
  I/O Channel 122
  Internal population 17, 66, 175–182
  Metal sizing 221–231
  Metal spacing 228–231
  Metal width 228–231
  Non-uniform 106, 115–124
  Notation 14
  Parameterization 64–75
  Pass transistor sizing 221–225
  Pass-transistor-switched wires 167–175
  Routing structures 208
  Single wire length 159–166
  Switch block 160–163
  Transistor sizing 138, 221–231
  Tri-state buffer 166
  Tri-state buffer sizing 225–228
  Two types of wire segment 166–175, 181–182
  Wire spacing 183–185
Row-based 14, 16
Roy, K. 16
Rswitch 94
Rutenbar, R. 25, 29, 90

S

Sangiovanni-Vincentelli, A. 23
Sechen, C. 24, 52
SEGA 99
Segment length 138
Segmentation distribution 16, 17, 66, 152
Shin, H. 21
Simulated annealing 22, 25, 52–55
Single Center Channel 120
Sink node 68
SIS 107, 130, 155
Slack 27, 32, 33, 34
Slack allocation 34
Source node 68
Spectral clustering 20
Speed 32, 142–147
Speed enhancement 79
Speed metric 17
SPICE 31, 134, 156, 217
SPLACE 98
SRAM cell 208
SRAM-based switch 12
SROUTE 98
Standard Cell 2, 3
Standard cell routing 30
Sun, Y. 18
Swapping 24
Swartz, J. 86, 100
Swartz, W. 24, 52
Switch block 14, 67, 72, 95, 160–163
Switch block flexibility 15
Switch block internal population, see Routing architecture : Internal population
Switch pattern 71

T
T, see Temperature
Tarrival 33
Tbuf,intrinsic 81
Temperature 23, 52–55
Tie breaker 44, 47
Tile 64
Tile-based FPGA 72
Time-to-market 3
Timing analysis 32, 67, 94
Timing graph 33, 94
Timing parameters 67
Timing-driven logic block packing 43–47
Timing-driven routing 26, 80–93, 101
Top/bottom pin positioning 110
TotalPathsAffected 46
Tracer 100
Track segment 109
Tracks per channel 67, 107
TransCount 133
Transistor area 132
Transistor level schematics 207–216
Transistor sizing 138, 155, 221–231
Trequired 34
Trimberger, S. 30
Triptych 24
Tri-state buffer 7, 16, 166, 210
Tri-state buffer sizing 225–228
Tseng, B. 15
TSMC 133, 154, 183, 209, 218
T-VPack 43–50

U
University of Toronto 7
Upstream resistance 85
Utilization 142

V
Validation 95–101
Vantis 5, 12, 14, 129
Versatile Place and Route 50
Vertical constraint 72
VPack 39–42, 48–50
VPR 50–61, 63–103
  Graphic visualization 199–203

W
W (Channel width) 14, 17, 64
Wilton switch block 159
Wire capacitance 81
Wire length 159–166
Wire model 31
Wire spacing 183–185
Wiring segment 16
Wmin 49, 130
Wong, D. 21
Wu, A. 99

X
XC4000 FPGA 29
XC4000X FPGA 151, 185, 189
XC5200 FPGA 39
Xilinx 5, 12, 14, 16, 29, 37, 39, 72, 100, 107, 110, 122, 129, 141, 151, 185, 189, 209

Y
Yang, H. 21

Since their introduction in 1984, Field-Programmable Gate Arrays (FPGAs) have become one of the most popular implementation media for digital circuits and have grown into a $2 billion per year industry. As process geometries have shrunk into the deep-submicron region, the logic capacity of FPGA's has greatly increased, making FPGAs a viable implementation alternative for larger and larger designs. To make the best use of these new deep-submicron processes, one must re-architect one's FPGAs and Computer-Aided Design (CAD) tools.

Architecture and CAD for Deep-Submicron FPGAs addresses several key issues in the design of high-performance FPGA architectures and CAD tools, with particular emphasis on issues that are important for FPGAs implemented in deep-submicron processes.

Three factors combine to determine the performance of an FPGA: the quality of the CAD tools used to map circuits into the FPGA, the quality of the FPGA architecture, and the electrical (i.e. transistor-level) design of the FPGA. **Architecture and CAD for Deep-Submicron FPGAs** examines all three of these issues in concert.

In order to investigate the quality of different FPGA architectures, one needs CAD tools capable of automatically implementing circuits in each FPGA architecture of interest. Once a circuit has been implemented in an FPGA architecture, one next needs accurate area and delay models to evaluate the quality (speed achieved, area required) of the circuit implementation in the FPGA architecture under test. This book therefore has three major foci: the development of a high-quality and highly *flexible* CAD infrastructure, the creation of accurate area and delay models for FPGAs, and the study of several important FPGA architectural issues.

Architecture and CAD for Deep-Submicron FPGAs is an essential reference for researchers, professionals and students interested in FPGAs.

ISBN 0-7923-8460-1

9 780792 384601

SECS497
0–7923–8460–1