cadence®

# Eliminating Routing Congestion Issues with Logic Synthesis

By Mike Clarke, Diego Hammerschlag, Matt Rardon, and Ankush Sood

Routing congestion, which results when too many routes need to go through an area with insufficient "routing tracks," can cause devastating delays to a project schedule. Logic design teams can avoid this problem and achieve greater success by adopting a physically aware and congestion-aware synthesis methodology that reduces iterations between front-end and back-end teams and positively impacts die size and schedule time.

## Contents



The following scenario has happened to a lot of logic designers; has it happened to you? You work hard to design a chip, make sure it meets the specifications, verify it, and hand it off to be physically implemented. You go on your merry way, planning your next project, when your phone rings in the middle of the night. It is the physical implementation team telling you that your chip cannot be routed because of congestion issues. And the chip might not be able to meet either the schedule or the spec. This is every chip designer's nightmare!

## What is Routing Congestion and Why Should You Care?

Simply stated, routing congestion occurs when too many routes need to go through an area that does not have enough resources—or "routing tracks"—to accommodate them.

Process geometries below 65nm enable more commonplace use of "high-density cells," whose cell heights dictate a decrease from 10-12 routing tracks through them to just 9. This means there are even fewer routing resources per cell, which will make the congestion problem even more acute. Not only that, but the use of pins on the "metal 2" layer is becoming more common, providing easier pin access but decreasing routing resources. And more metal resources must be dedicated to the power grid for higher power densities, higher frequencies, or complex power architectures such as multi-supply voltage or power shutoff—all very commonplace occurrences today. This means fewer resources for signal routes and a greater probability of congestion. And since higher speed designs have more signal integrity constraints, such as extra route spacing or wide routes, this also reduces available routing resources. Of course, creating lower frequency designs does not eliminate congestion either, as a higher ratio of combinational logic to registers means more pins to route to per unit of area.

These all sound like physical design problems; however, they can negatively impact the overall success of the project. When there are not enough resources with which to route, some routes must be detoured around the congestion.

Given the delay impact of wires in today's process geometries, modern timing-aware routers will try as much as possible to detour only those routes on non-timing-critical paths. But this is not always possible. And given the prominence of physical wire delays in today's geometries, it is likely that a non-critical path becomes critical after its route is detoured. Further compounding the problem, physical tools tend to buffer those detoured routes in an attempt to speed them up, and this can create further congestion.

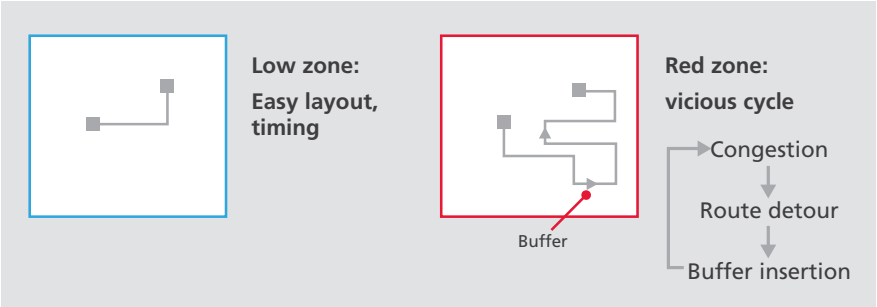


*Figure 1: Vicious cycle of congestion*

The unfortunate effect of this is a non-linear increase in routing difficulty. Just a small increase in congestion can cause this, and because of this it is often called "the red zone."

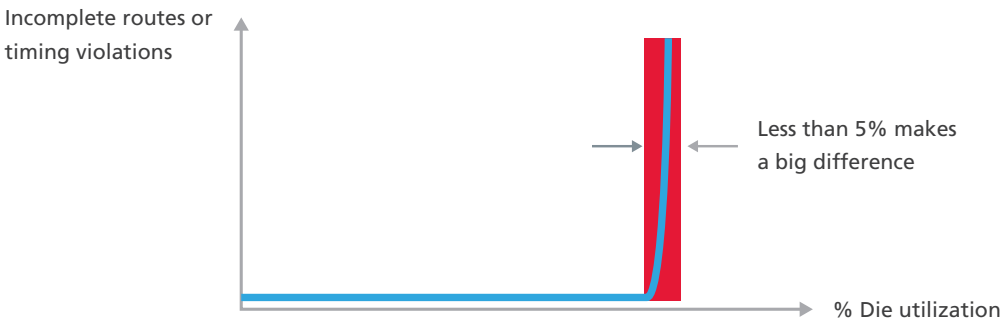


*Figure 2: The red zone*

So congestion creates more timing failures than the logic design team would anticipate because the only reason these paths fail is due to long physical wires that result from detoured routes. This is difficult for the logic designer to fix. And because the physical design team is not as familiar with the design and its timing requirements, fixing these issues takes time during physical implementation, which is often on the critical path of the chip's schedule. Thus, the project's overall schedule can be negatively affected by something that is beyond the logic designer's control.

## What Causes Congestion?

Before we discuss how to fix congestion, we need to understand what causes it. Unfortunately, there is not one single cause to address. There are actually a number of causes across two major categories: global congestion and local congestion.

### Global Congestion

**Global interconnect congestion.** This occurs when there are a lot of chip-level or inter-block wires that need to cross an area. For instance, interconnect between cells and I/O pins or memory ports will be very dependent on both the floorplan as well as where those cells are placed within the floorplan. Global interconnect congestion can occur even when there is low placement density—in fact, in some cases low placement density can even cause congestion because of the need for long connections and additional buffering. Finally, chips with a limited number of routing layers for cost reasons can also cause global congestion. All of these reasons are why it is so important to use the production floorplan and legalized production placement.

### Local Congestion

**Floorplan congestion**. This occurs when the floorplan has macros and other routing blockages that are too close together to get enough routes through to connect to the macros. For instance, the congestion can occur in slots between memories or around corners of memories. Identifying this type of congestion obviously requires that a production floorplan be used as an input.

Here is an example of floorplan congestion:

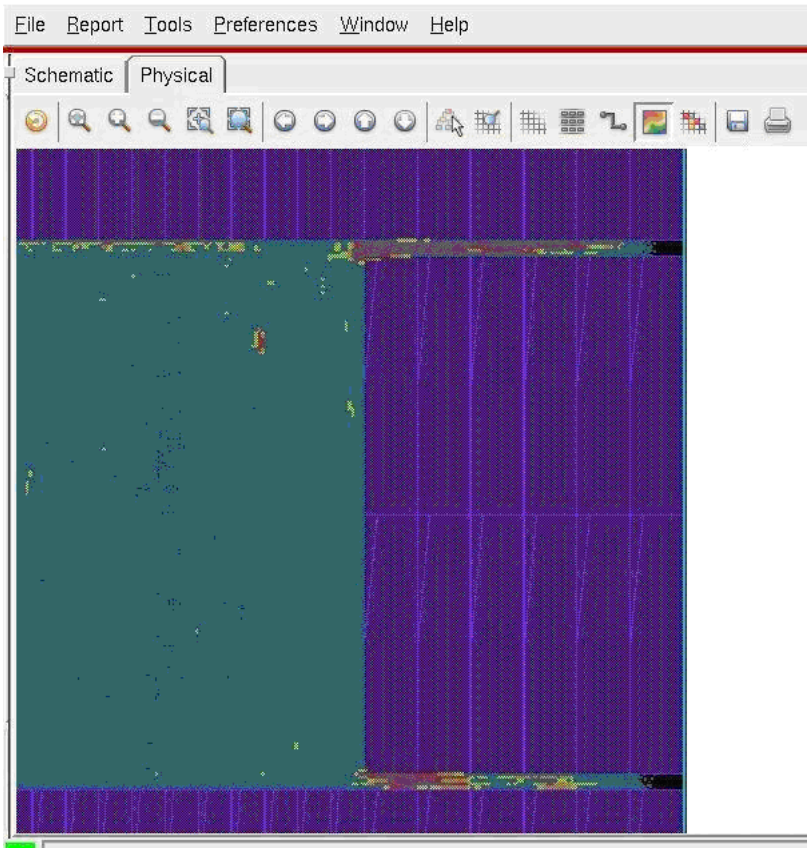


*Figure 3: Floorplan-induced congestion*

1. Placement density congestion. Placement utilization is basically how densely the cells are placed. When there are too many cells too close together (high placement utilization), then routing all the connections between them creates congestion. This can be the result of the entire design's utilization being too high, or perhaps it is localized to a block or region. Often this happens when timing is tightly (or too tightly) constrained, causing

timing-driven placement to place these cells closer together. Even more likely is that the netlist was poorly constructed. In other words, the netlist was created either without taking wire impact into account or was optimized with excessive incremental buffering and sizing. Identifying this type of congestion is dependent on the quality of the netlist, a production floorplan, and production placement.

2. Logic-induced congestion. Logic structure and cell selection can create congestion. A high amount of connectivity contained within a small physical area can overwhelm available routing resources. A common example is an extremely large multiplexor. There are just too many connections that need to be made in too small a space. Another example is high pin density (pin count divided by total cell area) caused by using smaller cells that require more routes to pins in a small area, such as decomposed complex cells or even low-drive cells.

Here is an example of logic-induced congestion:

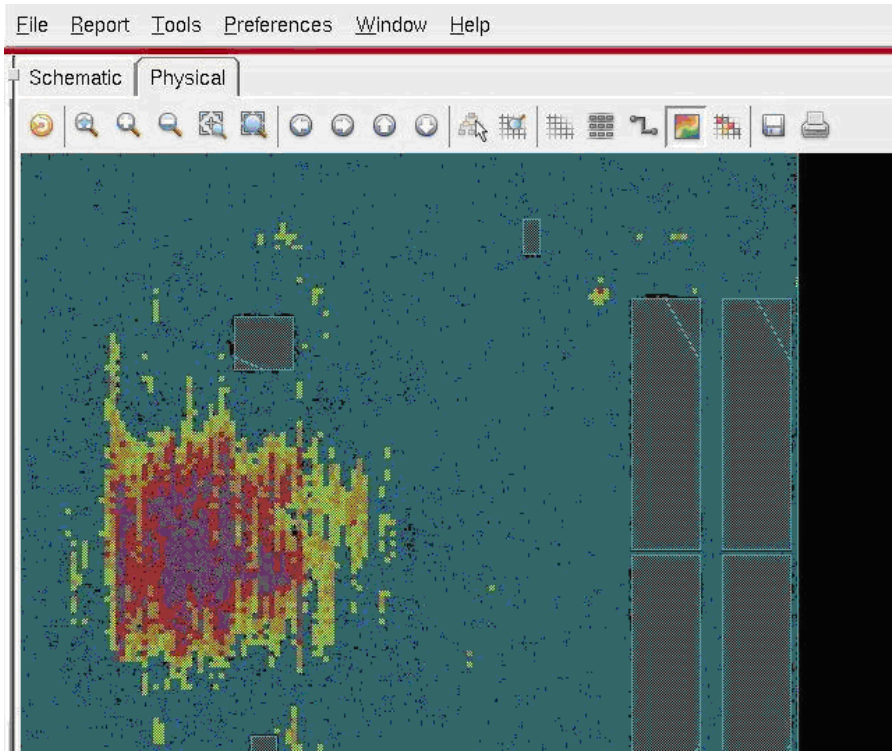


*Figure 4: Logic-induced congestion*

The challenge with logic-induced congestion is that it is often context-sensitive—a cell might be chosen for performance, area, or power reasons and might be fine for congestion in one context, but the same cell might increase congestion in another context. This is why it is critical to consider congestion in conjunction with performance, power, and area.

As you can see, the causes of congestion span logic design and physical design. This is what leads to a lot of the misunderstanding over whose fault it is when a chip cannot be implemented on schedule. Fortunately, this can now be avoided by analyzing the cause in physically-aware synthesis tools before handoff to physical design, and taking the appropriate action.

## Fixing Congestion

### Analysis

Fixing congestion starts with identifying that congestion is the problem and then determining the root cause. For the logic designer, this will require using a synthesis tool that can predict physical issues. More specifically, this requires high-quality production placement in the production floorplan—congestion is entirely dependent on the floorplan, placement quality, localized utilization, and power grid effects on the die. For instance, are there enough routing resources around the macros? Are there placement regions that must be obeyed, such as in the case of

power domains? Are there placement or routing blockages that must be obeyed? How densely can the logic be placed? How much buffering has to be performed on long wires in order to meet timing? Is there room to place these buffers in the optimal location? Is there space to upsize cells in their current location? It is important to model these possibilities as accurately as possible.

At the same time, if the goal is to provide useful feedback to logic designers, then the environment must be usable and understandable to a logic designer. In this case, all analyses must be performed within the existing logic synthesis environment. Once this is performed, quality of silicon (performance, power, and area measured with wires) can be determined and reported within the synthesis tool. At this point, if there are timing violations, the logic designer will begin to analyze the violating paths. In wireload-based synthesis, the wire delays on the violating paths will be proportional to the amount of fanout from the driving gates. In a physically-aware synthesis tool, there may be large wire delays where there is little fanout, or there may be a large amount of buffers on a path with little fanout—these will be long physical wires. If these long wires appear, they should be examined in a physical placement viewer to see why they exist. It could be a case of routes detoured around congested areas.

If congestion is identified as a problem, then all root causes must be understood before proceeding with the appropriate correction.

### Fixing global interconnect congestion

The approach to fixing congestion induced by global interconnect will depend on the root cause. In some cases, the fix will require restructuring of the logic to reduce interconnect. Other cases may require logic decomposition to spread logic along the route to reduce the need for buffering. Physical techniques may also be applied, such as incremental placement to move some logic closer together and spread other logic farther apart. For the more constrained cases, such as a crossbar switch that by nature has a high amount of connectivity, a combination of these techniques must be employed, which is why it is important to have adequate porosity, or pin accessibility, in the initial placement. This requires that placement and congestion be considered in conjunction with incremental logic optimization.

### Fixing floorplan congestion

The only way to fix this type of congestion is to address it in the floorplan. For instance, creating more space around the macros, creating placement blockages, or rearranging the macro placement or orientation are often solutions. This is best done by the physical design team, since they own the expertise in this area. However, it is important to note that if floorplan congestion can be identified as a problem during synthesis before handoff, then it can be fixed and the new floorplan accounted for during production synthesis. Just identifying the problem earlier can save headaches and delays later on.

### Fixing placement density congestion

There are a few ways to address this type of congestion, spanning synthesis and placement. Much of this can be performed inside a physically-aware synthesis tool.

For instance, a first step is to look at reducing the amount of cell area during synthesis. Are the timing constraints severely over-margined for performance? This will cause an increase in area and hence placement density. Also, if synthesis was performed with inaccurate wire delay models (for instance, with wireload models), then it may make sense to remove buffers and inverter pairs before placement. Or in many cases where the congestion is very localized, simply applying incremental placement moves like spreading and morphing can ease congestion.

If these types of fixes cannot be performed, then placement utilization should be assessed. Decreasing the overall target utilization for a chip is often the least desirable approach, since that implies that the die size will need to increase. It might also create a lot of unnecessarily long wires. A good first step is to try to decrease target utilization only for the specific region(s) where congestion exists. It is likely that this will require the specified region(s) to grow, which will cause increased utilization in other regions. If the other regions are already densely placed, then this could just move the problem. Utilization-based approaches can be performed in physically-aware synthesis tools, and physically-aware synthesis can help guide physical domains to the optimal size in context of the chip, but it is best to consult with the physical design team when attempting to use them.

### Fixing logic-induced congestion

Logic-induced congestion is caused by cells with a high amount of connectivity. The classic "512:1 mux" is a good example. In the case of the mux, congestion can be fixed either by restructuring the RTL code or by directing the synthesis tool to break up mux structures during optimization. These types of complex structures are advantageous for minimizing area, but if they are physically placed in a high-density area or if a lot of their connections need buffering, all that connectivity can cause routing congestion. Because these structures are useful for saving area, it is not desirable to avoid them entirely, which makes it even more important that the synthesis tool creating the logic structure also be aware of these physical issues.

The logic-induced congestion issues that are caused by the need for too many routes to pins in a small area—or high pin density—can be fixed during cell mapping. For instance, increasing drive strength will increase the cell area but decrease pin density (or increase pin accessibility). Replacing complex cells with more dispersed yet less complex cells can reduce pin density. Of course, reducing net crossing should also reduce congestion, due to less need for routes. Avoiding decomposition of an XOR cell into an AOI and NAND will also help reduce congestion. Of course, not all of these techniques are beneficial in areas free from congestion problems, so they should be either applied on a per-region basis or dynamically within the physically-aware synthesis cell mapping process.

## Preventing Congestion

Preventing congestion entirely is a difficult proposition due to its localized nature. It would be easy to prevent if you were able to globally reduce utilization or globally avoid complex cells. Since performing these tasks across the entire chip would cause a die size penalty and possibly even induce congestion, the more balanced approach would be to utilize congestion-aware synthesis to automate a lot of the steps described earlier.

The first requirement of such a synthesis tool is that it be able to model and measure congestion dynamically, so that it can make good decisions during the optimization process. This model needs to be updated with every change to the logic and placement, since congestion can sometimes be fixed in one place but moved to another.

Once this model is present, the tool can examine the various methods described above. Preventing floorplan-induced congestion is most likely out of its scope. But identifying congested areas and structuring logic differently, mapping to or avoiding certain types of cells in congested regions, and spreading the placement or re-setting the target utilization for congested regions, are all optimizations that can be performed inside synthesis. And modern global synthesis algorithms can even use congestion as a cost function when creating the initial logic structures, heading off some problems before they even occur.

The tools available to logic designers for addressing congestion have come a long way in a short time. But addressing congestion still has to start with analysis. Fortunately, this analysis can now occur directly within synthesis by starting with timing analysis as usual. Congestion is just another dynamic that must be examined and accounted for as part of this process.

Adopting a congestion-aware synthesis methodology is thankfully incremental over existing synthesis methodologies. And the rewards can be great—whether it means reducing iterations between front-end and back-end teams or simply saving die size or time in the schedule. Ultimately, a physically-aware and congestion-aware synthesis methodology returns control over the success of the project to the logic design team.