overlap of capabilities between synthesis and physical-design tools.

Synthesis Does Too Much

When originally designed, logic-synthesis tools needed certain optimization capabilities to help meet timing specifications. When synthesizing a structural representation (gate level) of a chip design from its RTL (architectural) representation, a logic-synthesis tool must choose a structure, often from several alternatives, that will meet timing and/or area requirements. The tool does this by optimizing the gate-level design to meet these specifications. The synthesis tool now comprises two sets of capabilities—**compiling** the gate-level design from an RTL representation, and **optimizing** the chip to meet design objectives—in the logical design space.

However, logic synthesis does a poor initial estimate of wire delay. When the physical-design tool takes the gate-level netlist from the synthesis tool, it must re-optimize the chip so that it meets design constraints in the physical design space. In other words, the P&R tools must "throw out" the optimization provided by logic synthesis and re-optimize taking into account the more accurate physical layout. This wastes time in the logic-synthesis operation (optimization dramatically increases run time) and actually hinders the operation of the subsequent physical-implementation tool.

Inserting a software tool between synthesis and physical implementation to more closely couple the logical and physical tools may provide some additional design optimization for small and medium designs (**Figure 2**). However, this added physical-synthesis tool does not eliminate the need to re-optimize large SoC designs in the physical-layout environment. In addition, a pre-P&R physical-optimization tool may also use a logic-synthesis core, thus just redoing the optimization done by the logic-synthesis tool—another waste of time.

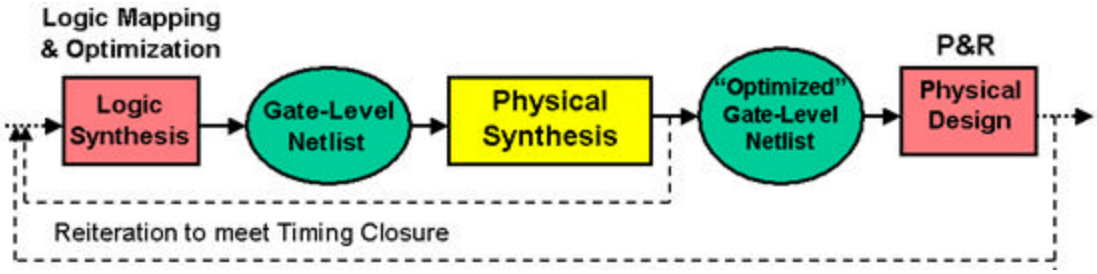
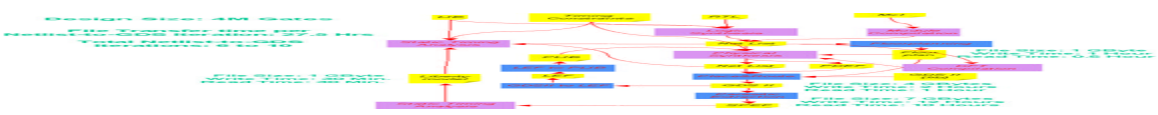


Figure 2: Chip design flow including a physical-synthesis tool

To illustrate the complexity and inefficiency of designing a chip using the current logic-synthesis/physical-synthesis/physical-implementation flow, a benchmark four-million-gate design took 26 weeks of design time. Each iteration through this flow requires around 20 intermediate files, comprising command files, scripts, data-translation files, and data files such as netlist, DEF/PDEF, SDF, SPEF, and PLIB. For a 4M-gate design, with 20-30 blocks, that adds up to several hundred files. Adding to the complexity of the accompanying data management are tasks dealing with directory structure, version control, user environments, hardware platform, and design network.

Parasitic extraction on our 4M-gate design creates a 7 Gbyte file that takes 12 hours to read out from an extraction tool and 10 hours to read into a static-timing-analysis tool. A complete iteration through the RTL-to-GDSII flow with commonly used point tools requires over 27 hours of run time and around 15 Gbytes of disk space for all the necessary intermediate files (**Figure 3**). Finally, to reach timing closure you typically need between 6 and 10 full synthesis/physical-implementation iterations. You can see that a lot of time and effort are wasted due to an inefficient front-end/backend handoff. There are no advantages in having such long file I/O times and large file transfers—they don't add anything to design quality.



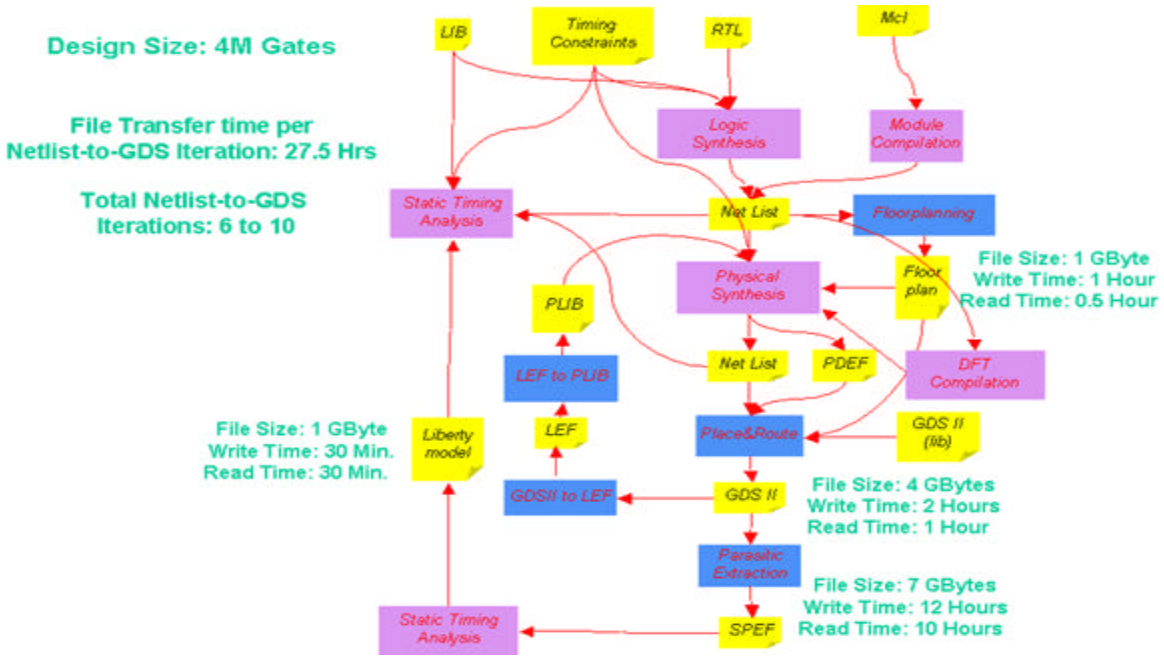


Figure 3: Current design flow is marked by large file sizes, long read and write times, and multiple synthesis/P&R iterations

Another problem with a physical-synthesis tool following logic synthesis is that such a tool is usually concerned only with timing specifications or, at most, also with some types of crosstalk optimization. Physical synthesis does not adequately address crosstalk and signal-integrity problems, since these problems are very layout dependent and the physical-synthesis tool does not have knowledge of the final chip layout. Furthermore, physical-synthesis tools do not address problems caused by interconnect inductance and do not understand the complex layout design-rules associated with complex SoCs. Even with physical synthesis, design issues such as these result in difficult-to-solve front-to-back design problems.

Size Is Important

Besides the issues already described concerning traditional synthesis to physical-layout handoff, another problem with popular synthesis tools concerns tool capacity. Logic-synthesis design-size limitations limit the size of the design that this type of tool can handle. Large chip size, even down at the few-hundred-thousand gate level, requires you to partition the design into several blocks, with each block within the capacity limitation of the logic-synthesis tool. Current SoC sizes at 10-million-gates and higher may require dozens of partitions, leading to problems reconciling the timing budgets between these blocks when considering overall chip timing. This is true even when the design includes several pre-designed silicon-IP cores.

Design partitioning requires guard-banding performance. This means that you must over-constrain interfaces to allow back-end tools to account for inaccuracies in front-end tools. You also have to define "registered" interfaces to allow constraints to be set—this results in extra logic in the design. In addition, you must derive the constraints and keep them consistent with any change in design-block constraints.

The need for a large number of design partitions places an additional resource load on both the design team and on logic-synthesis and other design tools. Partitioning adds complexity to resource allocation for the various serial and parallel design activities, and in accounting for the interdependencies between these activities. Reducing the number of partitions decreases the cost of design development and EDA-tool licenses, reduces time-to market due to decreased tool run times and chip design time, and improves quality of results.