

FEW DESIGNERS WOULD DEBATE THE ADVANTAGES OF USING A HIERARCHICAL-DESIGN APPROACH FOR DEVELOPING A COMPLEX CHIP.

Divide and conquer complex chip designs

HISTORICALLY, DESIGNERS have used a hierarchical approach to chip design—breaking the chip into pieces, or blocks—to extend the capacity of design-automation tools. Adopting a hierarchical approach has the advantage of enabling concurrent RTL (register-transfer-level) and physical design, because physical design can start before the netlist is complete. It also allows the use of multiple power regions on a chip. In addition, hierarchical design helps to contain last-minute design changes to local blocks (see sidebar “Glossary of terms”). However, it comes at the cost of much higher project complexity (multiple place-and-route jobs) and loss of optimal results (larger area and lower timing performance). Despite these drawbacks, designers of complex chips, such as graphics processors and microprocessors, have developed methods that exploit the advantages and minimize the drawbacks of hierarchical design by adopting procedures and a design-tool suite that can successfully reassemble the blocks and complete the design for a complex SOC (system on chip).

A hierarchical-design methodology and the tools it uses must be able to support timing and area optimization. The tools must view the design in a “full-chip” context to ensure better timing closure than what you can achieve when you design the separate blocks in isolation. Such design involves adequate repeater placement along with matching interconnect and cell delays to meet setup-and-hold and clock-skew specifications. In addition, a hierarchical-design physical-implementation tool should minimize or eliminate top-level wiring channels, which can increase chip size by 20% or more over the area of a flat design. The physical-implementation tool must also be able run reasonably fast, allowing you to iterate and reopti-

mize as needed to meet design specifications.

A tool suite that can simultaneously deal with both the hierarchical and flat aspects of a complete chip makes hierarchical design viable for any SOC-design team. The tools need to be able to work with global-chip features, such as power distribution, top-level clock distribution, and top-level repeaters, as well as with elements within blocks, macros and cells on a level below the top level (see sidebar “Hierarchical-design styles”).

HIERARCHICAL-DESIGN BENEFITS

Improved handling of the capacity limitations of EDA tools when dealing with multimillion-transistor designs is a major benefit of hierarchical design. Working on the chip’s constituent blocks in parallel allows multiple computers and designers to simultaneously work on a chip.

But, increasingly, other advantages are beginning

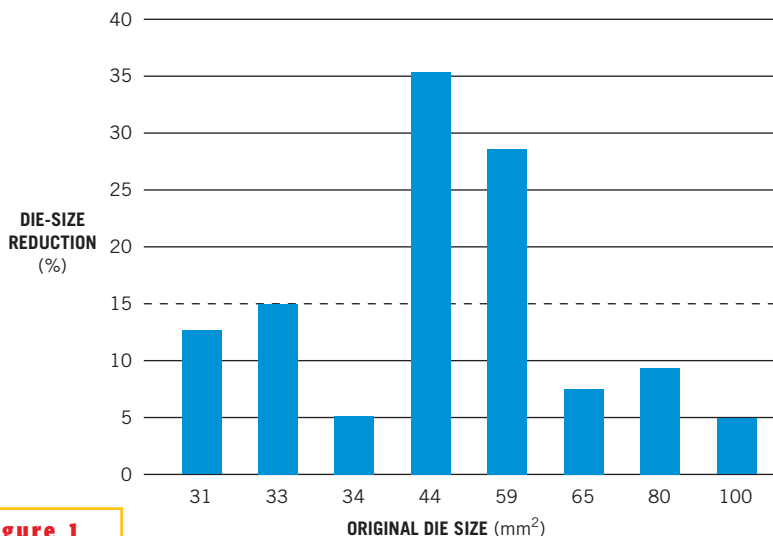
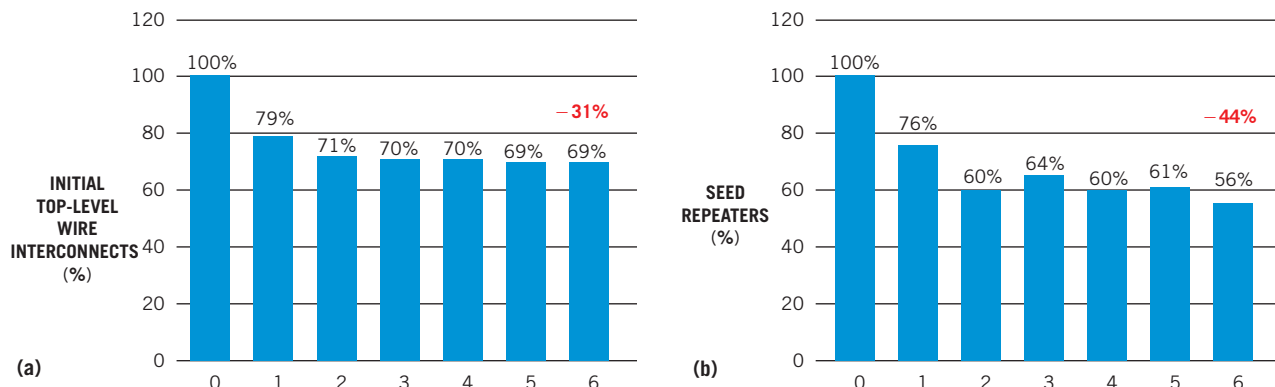


Figure 1

Hierarchical back-end chip design using true abutment eliminates top-level wiring channels, reducing chip area by an average of 15% across eight benchmark chips.



Graphs show the convergence behavior of the design-aware algorithm on reducing top-level wiring (a) and interblock repeating (b). Note the significant reduction achieved in just two passes of the algorithm.

to dominate. A hierarchical approach enables concurrent RTL and physical design. Concurrent design greatly accelerates chip development, because you can at an early stage uncover the physical design constraints, and the RTL team can correct them. With hierarchy, the netlist generally matures faster in some blocks than in others. You can floorplan these physically mature blocks and model the less mature blocks to obtain a better idea of the final chip's performance. You can also uncover design issues much earlier than in a flat design flow, which requires you to wait until the end of RTL development to get started.

Smaller block netlists afforded by hierarchical design have another advantage: They allow EDA tools to do placement and routing with a higher QOR (quality of results) and greater reliability with fewer core dumps than large netlists (Reference 1).

Hierarchical-design methods improve controllability. You can tweak design constraints, tool parameters, and the netlist to accelerate design closure by optimizing tool settings and putting extra time into physically floorplanning the blocks and the chip. This ability is key in tape-out, when there are always changes to the "final" netlist, usually due to system verification uncovering a design flaw. As the netlist matures, hierarchy creates an environment in which chip assembly is more deterministic. For example, suppose you need to resynthesize a block at the last minute before tape-out. In a flat design, this task would be impossible, because there is no way to remove and insert the new netlist for this block. With a hierarchical design, you can

easily accomplish such a change by redesigning the single block and keeping the same pin locations and other external constraints. This method allows you to make late changes to a portion of the chip with minimal impact on the rest of the design.

Finally, breaking the design into blocks in a hierarchical methodology also benefits design reuse. As you design and ver-

ify blocks, they become available for other designs, because you can now consider them hard-IP (intellectual-property) cores.

The biggest obstacle to successful hierarchical design is the "horizon effect," whereby block-design methodologies can prevent propagation of cross-hierarchy design parameters. It results in sub-optimal block design, unacceptable de-

GLOSSARY OF TERMS

Block: Representing a level of hierarchy below the top level of a chip, blocks comprise silicon cores and groups of abutted standard cells (a macro block) that constitute some function. EDA tools automatically place and route blocks, assembling the chip's physical database.

Cell library: A compilation of standard cells, hard-IP (intellectual-property) cores, and other macro blocks that comprise different functions within a library that a layout tool uses to construct a chip.

Channels: Gaps between the rows of cells of a standard-cell block that contain the intercell connecting wires.

Hard IP: A placed-and-routed block representing a function, such as microprocessor core, RAM, PLL, or ADC. Today's EDA tools cannot automatically place hard-IP blocks with good quality of return without the addition of a floorplanning operation.

Macro block: A block of standard cells and hard IP that has previously been automatically placed and routed, turning the macro block into a higher level hard-IP block that can go into a cell library.

Pure feedthrough: A signal wire that cuts through a block and needs no repeater or other logic to maintain signal integrity.

Repeater: A buffer added to overcome the resistance of long wires, thus sustaining the integrity of a signal that wire carries. A long net requires a repeater approximately every 1.5 mm in 130-nm technology.

Repeater feedthrough: A signal wire that cuts through a block and is long enough that the netlist for that block includes a repeater for the wire.

Standard cells: Low-complexity logic cells representing functions such as NAND gates, flip-flops, and buffers. All the standard cells for a design are of the same height, with I/O connections on the top and the bottom of each cell. Power connections are in common locations on both sides of the cells, allowing EDA tools to automatically place them, with power and ground connectivity assured by cell abutment. Intercell connections are made with wires running through channels between the rows of cells.

Top-level netlist: The top-most level of the physical-design netlist.

sign trade-offs at chip assembly, or both. Traditional design tools cannot “see” across block boundaries. Without this cross-boundary view, the tools cannot do full-chip optimization or correct potential problems resulting from QOR, structural, and flow issues.

QOR problems include global wiring congestion, global or lower level timing errors, area increase due to global wiring channels, poor block partitioning, and floorplan limitations on block shapes. Structural issues include DRC (design-rules-checking) problems at block boundaries, creation of global power and clock-distribution networks, global repeater insertion, and timing and routability constraints on block partitioning or shape. Among the flow problems are overall chip integration, data management of multiple-block design and verification involving people in of-

fices around the globe, and dealing with necessary design iterations to meet specifications.

USE THE RIGHT TOOLS

Implementing a design with a true-abutment methodology requires design tools for floorplanning and physical implementation with several unique attributes. A design-tool suite that supports hierarchical chip design must be able to look at both hierarchical and flat aspects of the design and successfully deal with the structural, flow, and integration issues that go along with such a design methodology. Such tools must be able to deal with several issues that hierarchical design introduces by supporting the optimization (eliminating wiring channels and minimizing top-level repeaters and wire length); fast builds (to get fast place-and-route feedback on design decisions);

and inevitable changes to netlist, IP, floorplans, and pad elements.

Wiring channels between blocks in a traditional hierarchical design increase not only chip area and signal-integrity risks, but also signal latency, resulting in a reduction of maximum on-chip clock rate. Designers need tools that construct the top-level chip, comprising individual blocks, without channels. Such tools must do resource-allocation optimization, including timing budgets and setup-and-hold timing across block boundaries, such that the final chip has the same area and signal-flow characterizations of a flat design (**Figure 1**). In addition, the tools should be able to work with commercial place-and-route tools executing multiple place-and-route operations in parallel to allow a designer do full-chip place and route in a reasonable time span.

HIERARCHICAL-DESIGN STYLES

Reviewing the several ways to physically build a chip reveals that one—true abutment—is best suited to dealing with the physical and performance complexities of a leading-edge SOC (system on chip) when you implement it with the correct design tools.

Conceptually, flat design is the easiest style for a chip designer. The entire netlist is at a single hierarchical level (the top level). All the library cells are within this top level, and placement and routing occur monolithically. The designer and design tools need not deal with any boundaries between the various design blocks, thus they need not contend with timing-partitioning issues. However, flat design leads to problems with the runtimes of the various EDA tools you use during place-and-route operations, tool-convergence repeatability, local QOR (quality of results), and incomplete netlists.

In classic-channeled design, the top-level netlist usually contains the cells that go into the chip’s pad ring along with some

other library cells. Pure classic-channeled design has no logic in the channels and uses preferred routing directions in the channel. This type of design is an artifact of the time when designers had less than four levels of metal for interconnecting blocks and cells on a chip. You can use the pad ring as one large macro overlay, use multiple macros for each side of the chip, or do some other pad-cell partitioning. Today’s designers rarely use classic-channeled design, because its structural issues are difficult, and process technology with six or more metal layers makes it unnecessary.

Top-flat design is the most common hierarchical style today. Engineers always route the wiring channels with standard routing directions. This level of design may include a large number of standard cells. If the number is too large, the EDA-tool runtime may become excessive. If this situation occurs, designers create blocks of cells, adding another level of hierarchy, to reduce the total number of cells

and blocks at the top hierarchical level. Placing blocks and other hard IP (intellectual property) at the top hierarchical level seeds the top-level standard-cell-placement process.

You use almost-abutment chip design on small netlists with only two to three blocks, in which the design connectivity and partitioning avoids a large number of feedthroughs (a “friendly netlist”). Netlists with 10 or more blocks use special EDA tools and netlist “tweaking” to add more repeaters in blocks, allowing more cell abutment. All or most connections at the top-level of the netlist then appear to go from block to block, making the entire netlist friendly for floorplanning.

Place-and-route tools use small channels to route a few “unfriendly” signals around intervening blocks. Clock distribution is typically difficult. Engineers accomplish it within the channels or by “hacking” all the block netlists. Users must partition other global structures besides repeaters and clock-distribution

hardware or make sure that they are at the top level.

In true abutment, which targets complex chips comprising many blocks, all blocks are fully abutted, and no logic or layout remains at the top level of the hierarchy. Because all functions reside within blocks, the method simplifies chip integration; only blocks are present, and there is no top-level glue logic. All block-to-block interconnects use repeater feedthroughs, pure feedthroughs, or both through intervening blocks. Appropriate tools automatically share all metal resources for local and global routing, such as clock trees and power networks. Critical signals can proceed directly across the chip, repeating their way through intervening blocks as if the design were flat.

Block partitions that would be difficult to handle in other hierarchical methodologies are easy with true abutment. Blocks with high terminal counts and several thousand feedthroughs present no significant problems for chips designed with this methodology.

High-quality global routing to determine pin locations is necessary to minimize block congestion and to optimize timing and power distribution. A tool suite targeting hierarchical design employs a multipass-design paradigm, using a global router that can simultaneously open previous versions and the new version of the design. Because initial pin locations heavily influence block placement, a carefully seeded global routing prevents the placer from falling into local minima. Successive builds with feedback layout then “relax” to an approach that both minimizes global resources and produces the best block placement. Problems with timing or congestion that the block “feels” in its full production environment feed back, and you can then optimize them to solve the problem (**Figure 2**). This methodology is similar to one that a good designer would use—reviewing the most recent chip build and implementing changes to improve the design on the next build.

Allocation of clock-timing constraints

and repeater insertion are additional attributes of back-end hierarchical design tools. The designer specifies global (pin-level) timing specifications, from which the design tools generate block-level timing constraints for static-timing analysis and placement and routing. After each full-chip build, the tools should update these constraints to ensure full-chip timing convergence. As global placement and routing progresses, the tools must also recognize when to insert global repeaters to meet timing specifications. Inserting timing repeaters is a challenging task, because a design may need several thousand repeaters to meet timing constraints.

Successful hierarchical-chip design results from applying the right tools and design methodology to create a true abutment design. Such a design offers not only several performance advantages over a flat design, but also a geographically diverse design team whose members have different skills can more easily implement it. □

REFERENCE

1. Cong, J, et al, “Optimal Scalability Study of Existing Placement Algorithms,” Asia South Pacific DAC, January 2003.

AUTHOR’S BIOGRAPHY

Paul Rodman is chief technology officer and co-founder of ReShape Inc (www.reshape.com). He has more than 20 years of experience in CPU design and has participated in the architectural and physical design of the MIPS R8000 and Nintendo 64 processors. Before 1990, he worked at Multiflow Computer as an architect and designer of the world’s first VLIW (very-long-instruction-word) computer. He holds a bachelor’s degree in astronomy from the University of Massachusetts (Amherst) and a master’s in computer engineering from Carnegie-Mellon University (Pittsburgh). His leisure interests include travel and photography.

TALK TO US

Post comments via TalkBack at the online version of this article at www.edn.com.