Data Modeling and Convergence Methodology in Integration Ensemble

Lou Scheffer

Cadence, San Jose, CA, USA

Abstract

Integration Ensemble (IE) is a tool that enables teams of designers to go from RTL to GDSII. This paper discusses the choices behind the data model for such a tool, and the use models, methodologies, and features required for design convergence.

Introduction

We start with a quick explanation of what Integration Ensemble is and why it was built. Next, we describe a thought experiment that shows why advanced handling of hierarchy is required. Then we discuss how the use of hierarchy drives the data model. We discuss the use models and how they contribute to design convergence, and the new features needed to support this. Finally, we draw some conclusions about how flow tools differ from point tools.

What is IE, anyway?

Integration Ensemble (IE) is an integrated single tool that works helps the designer transform RTL to GDSII. It has three main goals:

- Make flows easier by building a single integrated product that can replace the collection of existing Cadence tools now required.
- Provide new features to make hierarchical design flows easier and more productive.
- Provide the tool for the digital part of the mixed signal flows.

Historically, customers used different tools for each step in the design process from RTL to GDSII. For example, they might use a floorplanner, a synthesis tool, a placer, a router, and a timing verifier. The tools were developed and sold this way (even if all the tools were from the same vendor), and the customers often preferred different tools for reasons of price, performance, or features (the 'best in class' approach).

The very existence of this conference shows the problem with this approach. Designers needed to learn several different tools. Subtle incompatibilities (particularly in timing) hurt design convergence. Users had to cope with multiple libraries and design translations.

IE was built with the goal of optimizing the flow as seen by the designer. This drove a number of technical decisions. First, we used a single, very high capacity database (Genesis) for all tasks (floorplanning, synthesis, timing, placement, routing, and signal integrity). This already shows a flow bias since such a combined database can always be outperformed by a task specific one. Second, we modified each task to use a common timing engine. Once again, this did not help the individual tools, but made the flow work much better. Finally, we replaced 6 different tool user interfaces with a single cockpit/GUI from RTL through place & route. This was a huge step in terms of making the flows easier. Finally, since the flow context of each tool is better known, each tool became simpler. The entire flow from RTL to GDSII now has less than half the menu options of the old floorplanner alone!

In contrast to previous Cadence products, the emphasis in IE is not on point tool features; we just tried to incorporate the best technology from tools such as PKS, Qplace, Wroute, IC Craftsman, CT-GEN, and the block placer. The new features that were added are there to help flows, not because they improve point tool operation. In particular, hierarchical flows have always been particularly difficult, and the new features and options were added to make this easier. This required a much-increased emphasis on constraints and budgets.

Although IE already handles many more steps in the design flow than its predecessors, it by no means handles them all. Additional flow steps may include:

- Inclusion of custom digital, datapath, analog and mixed signal blocks, and MEMs.
- Design specific techniques for test insertion and design for test.
- Functional verification.
- Chip finishing (adding alignment keys, scribe grid, tests structures, and other manufacturing necessities)
- Physical verification
- Manufacturing signoff, including PSM and OPC

In some of these cases, such as MEMs, IE will probably never handle the problem directly, but will be used as a sub-system to design the digital portion of the chip. In other cases such as chip finishing, IE will be extended to handle these flow steps.

Data Model

The big question is whether the data model is flat or hierarchical. Contrary to popular belief, hierarchical is NOT essential for CPU and/or memory reasons. Most CAD algorithms already scale as O(NlogN) or better, and since the machines available scale with problem size, we will probably always be able to do a design flat if we really want to. Furthermore, hierarchy induces an enormous number of complications. Never the less, we decided to do it anyway. To show why we'll imagine a thought experiment.

A Thought Experiment

Suppose you are a startup with the following requirements

- 250K lines of code to write
- 8 months to do it in, and get it right

Suppose you have a perfect flat tool for synthesis, place and route, that gives *optimum* results in *zero* run time. How do you organize your chip construction effort?

A normal coder generates about 6 lines per day of fully debugged code. As a startup, you hire only wizards who can write/debug 60 lines per day, by being exceptionally intense and working 16 hours per day. Now assume the whole team works 7 days per week for 8 months. In this time they can write and debug about 15,000 lines of code each. Therefore you need about 16 developers.

This is too many to manage as a single group, and there are too many people for each to know what everyone else is doing. The most natural organization is to divide into teams. But each team needs to know of other team's plans since some (important) things depend on whole chip organization - things such as power budgets, delays in long wires, signal integrity, and so on. Furthermore, the hypothetical synthesis/placement/routing tool, although perfect, still needs legal input, so each team will need a 'known good' version that the other teams can use while they build a better version of their own portion. Finally, the other teams will need to know how to interact with at least some of a given team's work. They will want this portion to be well defined and stable. This leads to the definition of interfaces and budgets. Finally, the flat tool will make it hard for each group to keep track of their own progress. Very soon, each group will define their portion of the chip, with interfaces, so they control their own progress. They have re-invented hierarchical design! *This conclusion holds even if the designers have a perfect flat tool available.*

We therefore conclude that hierarchy is neither for the benefit of the tools, nor because of their limitations. Hierarchy is for the benefit of the humans, not the machines.

Use Models for Hierarchical Design

From the previous section, it's easy to see that designers will need to use hierarchy. However, hierarchy can be used in many different ways. Two main variations are in the type of hierarchy that is used and the order in which tasks are done.

We decided we needed to support three types of hierarchy. The first is flat, or no hierarchy at all. This is needed for two reasons. First, for simple designs it is by far the simplest flow. Users would prefer not to face the complexities of hierarchy unless the design requires it. Second, flat design is needed to build the blocks even in a hierarchical environment.

The next hierarchy style is a pure block design. Here every piece of logic is assigned to one of the blocks. The blocks can then be designed independently. This is most useful on very large projects or those that have geographically dispersed groups.

Finally we have a mixed hierarchy. Here the vast majority of gates are assigned to one of the blocks, except for a small percentage that cannot easily be assigned to any block (this small percentage of gates may still number in the thousands, though – too much for any manual operations.) The user wants these gates sprinkled among the blocks at the top level.

On top of these three main hierarchy types, there are two task orderings that must be accommodated. In 'top down' design, the overall chip design sets the budgets for the blocks. These blocks are then built to meet the imposed budgets and then used to complete the chip. In the 'bottom up' flow, the blocks are built first. These completed blocks are then assembled into a chip. The chip design here must accommodate the block specifications.

Both top down and bottom up are needed in different circumstances. Top down is needed when the desire is to split a design problem across multiple groups, particularly where the groups are dispersed. Bottom up is need for IP blocks, since the blocks are completely designed ahead of time, and the global chip layout and interconnects must accommodate the already designed blocks.

Data Models for Hierarchical Design

The data models are defined by the need to handle the hierarchical design styles defined above, both bottom up and top down. Here are some of the problems faced by the users, and how the use of hierarchy impacts the data models needed:

▪Timing. The data model must support timing constraints that span block boundaries, and allow budgeting of global delays onto individual blocks. There must be an easy way for the user to specify the expected characteristics of blocks that are not yet built.

▪Parasitics. In a hierarchical design, once a block is built in isolation it must be incorporated into the chip. Signals entirely contained within a block can be reduced from parasitics to delays. Signals that connect to the pins of the block must be kept as detailed parasitics, since the rest of the net is not yet known and most reduction algorithms require the whole net before processing. Therefore, the data model must support a mixture of reduced nets and fully detailed nets.

▪Over the cell routing. If a global signal is routed over a block, perhaps with buffer insertion, then the block designer must accommodate the needs of the overall chip architect. These requirements can be treated as requests or commands, depending on the importance of the signal at the top level. Clearly the block designer must allow for the resources used for top level routing, and there is the possibility of interference between the over the block routing and signals inside the block. For all these reasons, the data model must distinguish routes assigned from above and routes with the control of the block.

▪IR drop. The IR drop allowed for a block is only a fraction of that allowed for the whole chip. If a block is built for incorporation into the top level, a simplified model of the power supply network for the block must be available to the top level.

▪Antenna rules. Depending on which is done first, the block or he global interconnect, the second task has limits on how much of each metal layer can be attached to the pin, and perhaps on which cells and drivers are allowed.

▪Power routing. The power structure (grids, rings, and/or stripes) is usually defined for the chip as a whole. Since this is very hard to change near the end of the design cycle, it's best to get it right early, before the detailed block design. So the individual blocks must be able to inherit the power supply structure from the overall chip design.

▪Multiple power supplies. Modern chips, particularly those that are power sensitive, typically contain many different power supplies, of the same or different voltages. This has many implications for the data model, particularly for timing, signal integrity, and output of the completed netlist.

Convergence Methodology

Better design convergence is one of the main reasons for an integrated tool, so we attack it on a number of fronts. The principles are very simple:

- Do the hard parts first
- Do it right, not do it over
- Prevent *AND* fix
- Suppress unneeded detail

Specifying the tool to work best with three particular hierarchical design styles makes this task much easier.

Do the Hard Part First

What *is* the hard part? Of course, you can always find the hard part simply by trying the design and seeing where you have the most trouble. An expert who has done a similar job already can really help you here, though.

The 'hard part' will vary from design to design, and so does the features needed to attack it. For example, in many chips the most difficult part is the global wiring. In this case the NANO flow [1] should be used, which does the global routing first. The pin positions on the blocks should be chosen to make the global routing easier, not to ease the block design problem.

Another possibility is that the 'hard part' is the design of a particular block. In this case we should do the hard block first, and let it set its own pin locations to where it thinks is best. Therefore a 'bottom up' flow is best, at least for the troublesome block. Furthermore, if the block is difficult, changes to the RTL may be required to enable it to be implemented at all. In this case, we strongly desire a tool that gives the RTL designer instant feedback about the physical effect of their decisions without waiting for a place and route cycle by another group. This drives the RTL designer to use a tool such as PKS as an accurate feasibility estimator, even if the exact results will not be used in the final design of the chip

A third possibility is that the 'hard part' is meeting a size, cost, or power target. These are properties of the whole chip, not just one block or wiring section. The best way to converge here is to start early with estimates for all blocks and update often as blocks progress. In this case it is crucial to run the real tool as early as possible, to make sure the estimates are reasonable. These tools include the detail router, synthesis, clock tree generation, and any other tool that can affect the desired global properties.

Do it right, not do it over

The worst convergence problems occur when the RTL seems OK but cannot be implemented with the desired performance. After many futile attempts at implementation, the task needs to go back to the RTL designer along with some hints as to why it was infeasible.

The best way to avoid this is to allow the RTL designer to see the implications of what they write. This means they should have access to a physically knowledgeable synthesis tool as they design. This allows them to check, as they go, whether their design is OK. Although it may sometimes be unneeded, it provides a lot more confidence.

Suppress Unneeded Detail

Chip design in general, and hierarchical design in particular, have an enormous amount of detail only a tiny fraction of which is useful to a designer at any given time. Two examples of this are global wiring budgets and consideration of DSM effects. Someone needs to worry about these problems, but if that person is the RTL designer the chip will never be completed. It is crucial that all these details be handled automatically wherever possible.

Every designer-entered budget number can make an otherwise easy design infeasible, so we need a budgeting and allocation strategy for all problems. Historically, this has been the biggest barrier to hierarchical design. A multi-million gate design may have tens of thousands of top level nets and pins. Each of these nets and pins need constraints if the block design is to be farmed out to separate teams, and a single bad number can make the block and/or top level design impossible.

The new features in IE are concentrated in this area. The NANO methodology provides one particular way to account for global wiring delays in the budgets. Shell/Core partitioning provides a way to tradeoff the budgets of the blocks against each other to arrive at a mutually acceptable compromise.

Although DSM effects are crucial, we cannot expect that RTL designers, for example, will be experts in inductance, or crosstalk, or antenna effects, (In fact, 90 percent of the people involved in the flow have no idea of what an ‘antenna effect’ is, other than that it is some ugly DSM problem that needs to be fixed.). These effects must be fixed as automatically as possible, and the portions of the effects that are of interest to RTL designers (such as timing degradation) must be incorporated in an easily understood manner.

Prevent *AND* Fix

Correct by Construction is great, but not always practical. For example, crosstalk control can be done in correct by construction[2], but these techniques that using nothing but prevention have a huge overhead.

On the other hand, *Construct by Correction* can yield unpleasant surprises. For example, widening a power line after final place and route is really painful, since many existing signal wires will need to be relocated, with probable timing, signal integrity, and routability impacts.

We need a strategy for each chip problem, with the appropriate features for fixing it. A sample of some problems and where they should be addressed in the design cycle for maximum productivity:

- Supply planning must be done very early, since it is very difficult to make the wires bigger later.
- Crosstalk glitch control can be left to post-routing. In general, circuits on the critical path have strong drivers and short routes as a consequence of timing driven sizing, placement, and routing. Most crosstalk glitches happen on side paths, and can be fixed after routing.
- Crosstalk delay must be anticipated during synthesis, since the router can only reduce, not eliminate, the effect.
- Antenna problems should be avoided by global buffering but also must also be fixed during detailed route.

A quick look at this list shows one of the main advantages of an integrated tool – a given user could discover by experience the best strategy for each problem, but a pre-built flow thinks about these problems in advance and makes it unnecessary to learn from experience.

Conclusions

The main conclusions are: hierarchy is needed for the humans, not for the algorithms. The need for hierarchical design (and the hierarchical design styles supported) determines the data models needed. Design convergence in flows requires both well-specified use models and explicit consideration in tool design.

References

[1] “NANO Project would reroute today's synthesis-to-layout flow – Cadence maps design overhaul", Electronic Engineering Times, May 17, 1999

[2] S. P. Khatri, A. Mehrotra, R. K. Brayton, A. Sangiovanni-Vincentelli, and R.H.J.M. Otten, "A Novel VLSI Layout **Fabric** for Deep Submicron Applications," Proc. DAC, 1999, pp. 491-496.