Hierarchical physical design—issues and methodologies

Steve Lloyd - July 12, 2013

In IC physical design, there is a tendency to focus on the synthesis and layout tasks, and to not give much consideration to the chip finishing tasks, at least not until the more pressing matters of timing closure and power analysis are in hand. This can be a dangerous practice to follow, and for large hierarchical designs there is a significant risk that issues identified during top-level finishing could require flow modifications at the block level, causing blocks to be re-opened and schedules to slip. Also, there is the wider issue of consistency across the blocks of library versions, flows, tools, and procedures, particularly where different parts of the design are farmed out to different design team members.

In this paper, we will examine some of the issues and methodologies encountered during the finishing and physical signoff stages, and discuss how some of the risks can be mitigated by forward planning and careful data management based on successful experiences of Synopsys Professional Services design consultants implementing large designs. We will also look at additional methods that can be used to give a high degree of confidence in the integrity and manufacturability of a large hierarchical design.

Hierarchical physical designs

For large designs, there are advantages to be gained from splitting the physical design process into manageable chunks, or blocks, that are then combined together at a higher level of hierarchy to form a larger composite block, or the top-level (Figure 1). This physical partitioning of the design is different from the logical hierarchy imposed by the structuring of HDL languages, although it will usually follow the same boundaries such that there is a mapping between logical and physical hierarchy. The partitioning of the design into physical blocks does require a good understanding of functionality, data flow and hierarchical design methodology, but is not itself a subject for discussion here. We are going to examine what happens towards the end of the flow when these blocks are brought back together and integrated into the final chip design, in preparation for tapeout.
One aspect of hierarchical design that can be easily overlooked is data consistency, i.e., ensuring that everyone and everything is using a common, consistent set of source data. A typical design might use five or six different standard-cell libraries, two or more pad libraries, many memories from more than one vendor, assorted third-party IP blocks, and in-house macros that may be separately built, such as processor cores and analog modules (Figure 2). Good data management would involve tracking of release versions, scripted flows for building tool-specific databases, and careful checking or results. Here are some pitfalls to watch out for:

1. Different tools use different views of the original data. For example, the physical layout of an IP block is often supplied as a GDS file, however this needs to be converted into a database format suitable for place and route tools. During tape-out, the place & route tool will write a full-chip GDS containing this IP, but how can we be sure that nothing has been lost during the translation process?
2. Different library data is used by different tools at different stages of the flow. The location from which data is loaded needs to be specified somewhere, maybe in a script, in a configuration file, or stored inside the design database. How can we be sure that a consistent, coherent set of data is used, i.e. that all tools reference the correct version of the IP data, and stored or cached references are forcibly updated when a new version is installed?

3. Hard-coded ROM programming is often only finalized near the end of the project, and may be delivered just days before tapeout. We need to be absolutely sure that this new data gets picked up during stream-out, and model (schematic/source) netlists are also updated to incorporate the new coding.

Flow and methodology

Flow and methodology

Good flow and methodology is the cornerstone of any good design, and is doubly important for hierarchical designs and multi-user projects. Good methodology can help to minimize all the issues identified above, plus many more, and reduces the overall risk level while improving quality and integrity. Methodology is encapsulated by flow, i.e., by the scripting and sequencing of tasks that need to be performed in a reliable, repeatable and deterministic manner.

A reference methodology for our chosen toolset should form the basis of our flow, but these are not production flows and are not specific to any particular foundry technology. Many design companies invest in developing their own production flow, but for design organizations that decide their resources are better deployed on design tasks rather than creating and maintaining a state-of-the-art design flow, a pre-validated and customizable production flow such as the one contained in Synopsys’ Lynx Design System (Figure 3) is available off-the-shelf.

Figure 3 The Synopsys Lynx Design System is a state-of-the-art, pre-validated, customizable design flow.

Whatever our choice, a proven flow is essential for reliable, high quality, deterministic results.
For large hierarchical designs flow management can be a challenge, with each block sharing the same core flow but with its own individual requirements and nuances. Below are some common problems:

1. Several customized versions of the same script exist for use on different blocks in the design (e.g. floorplan script), but how can a common project-level update be applied to them, and how do we associate the correct version with each block and/or designer? A revision control system such as CVS or Perforce is very helpful here, and if used properly will help to prevent one person’s update from breaking another’s flow.

2. As new tool versions and database schema become available throughout the life of a project, we need to ensure consistency between blocks and between designers. If a specific block requires the use of a specific tool version, then this needs to be configured into the flow for that block. Remember that as designers come and go, different staff members may pick up a block to perform ECO operations, etc., and they need to work in the same design environment as the original designer. The use of modulefiles (or similar) is highly recommended to ensure all team members access the same tool versions and share the same common data.

3. Physical signoff tools use rule sets to check that designs are manufacturable and functionally correct. Some 3rd party IP blocks require specific option configurations in order to satisfy these signoff checks, and these configurations therefore have to ripple-up through the hierarchy. Ultimately, a single common set of rules and options must be used for full-chip verification, so this configuration needs to be identified as early as possible, and applied to the development of all blocks and top-level.

Figure 4 A typical standard cell contains structures not fully contained within its cell boundary.
Physical layout

The requirements for physical layout of hierarchical blocks are generally well understood. The use of routing keepouts around the edge of blocks ensures that spacing violations do not occur to objects in adjacent blocks, and also minimize the risk of signal crosstalk. Dummy fill is inserted at the block level so that any impact on timing can be resolved, and any local metal density issues addressed. There are also some not so obvious pitfalls:

1. Standard-cells, and maybe some macro blocks can contain structures that fall outside of their cell boundary (e.g. NWELL regions) (Figure 4). Placing cells coincident with or very close to the block boundary can lead to issues where these structures violate against structures in adjacent blocks or at the parent level. These violations may go unnoticed until full-chip physical verification is run, at which time it is costly to have to open the block(s) again to resolve the problem. We need to ensure that all structures are fully contained within the cell boundary.

2. Physical verification at the top-level works across block boundaries and some checks will aggregate the data as a flat view. For example, density checks will operate inside a check-window that moves across the chip in fixed increments from the origin, but it is very unlikely that this will align with the origin for each of the blocks, so new violations may result. In some cases the problem might only be fixable inside the block, resulting in a schedule slip.

3. Metal density gradient violations can occur between areas with dense power structures and those with sparse metal on those layers. Often these violations are ignored or not seen during early development because dummy fill is only added late in the flow; however, this can be costly to fix late in the project lifecycle.

Additional layout data integrity checks

Additional layout data integrity checks

We can now look at some methods that can help us minimize the risks inherent in a large hierarchical design.

Library QA checking

Quality assurance checking of incoming libraries can be performed to different degrees, but a minimum set of checks should include running full DRC/LVS checks on the data, in addition to some general library checking commands within the layout tools. Ideally these DRC/LVS checks should be run on the supplied GDS, and also on a GDS file streamed out from the IP library database using the same stream-out options the final tapeout would use. This gives us the additional assurance that the quality and integrity has not been compromised during the format translation, particularly as a result of any layer-mapping operations or other transformations.
**LVL checking**

LVL (layout versus layout) allows two GDS files to be compared (XOR or NOT) on a layer-by-layer basis (Figure 5). This is extremely useful for validating IP blocks in the streamed-out GDS against their original (golden) GDS data. Not only does this ensure that nothing has been corrupted in the block, but it also confirms that the specified IP version has indeed been implemented. This can be invaluable where ROMs are concerned, since an essential part of any tapeout checklist should be to confirm that the correct ROM code is used. Do not rely upon LVS because LVS checks the layout against the schematic, and any error in referencing the wrong ROM layout during stream-out is likely to also result in the wrong ROM schematic being used for LVS, which will then fail to flag a violation.

Some tools also offer a layout integrity management system. The principle is that a database of checksums for known good IP layouts is created, and subsequent design databases are validated against this database as part of physical verification. The effect is very similar to running LVL except that the golden reference is now the checksum database, and more informed reporting is possible.

It is recommended that all IP or 3rd party blocks be verified against the golden reference using LVL, or a layout integrity management system.

**Common library reference data**

With multiple users working on multiple blocks, it is important to ensure that a common set of references is used for linking to cell and IP libraries, and that references to different views of the same IP are consistent. Since each IP has several data files associated with it (timing models, netlists, layout data, schematic views, etc.) then these files must all be accessed from the same base location. Human error is minimized by confining all such references to a single shared configuration file that is carefully maintained and checked (Figure 6). The use of common path variables within this file could further reduce the risk of error, and a revision control system can be used to synchronize this file between users, while also allowing for the local differences necessary during the development stages of a project.
A single, shared configuration file can be easily maintained and checked. For hierarchical design blocks, a similar approach can be followed according to project requirements and flow methodology. As with IP blocks, it is important to ensure that a pointer to the latest good release of each block is maintained.

Some database formats (e.g. Milkyway) store reference library paths internally, so it is no longer sufficient to rely on the above common files being up to date. It becomes necessary to run additional checks prior to tapeout to ensure that all blocks are referencing the correct libraries, and that hierarchical design block references are also correct. One method for achieving this is with a script that recursively scans the data hierarchy from top to bottom, checking all library and block references, and following links to the other hierarchical blocks. A report can then be generated for all the inconsistencies.

*Chip finishing – dummy fill*
While it can be efficient from a file size point of view to insert dummy fill as array structures, these are not ECO-friendly and can cause hierarchical design problems in terms of visibility down the hierarchy, and also in tool usage. Effectively it is adding additional hierarchy into an already hierarchical design, for little gain. Consider also any requirement for dummy via structures and the picture gets even more complicated.

Dummy fill should be a bottom-up process, with the blocks at the bottom of the hierarchy being filled locally, and exclusion regions added over these blocks at the next level up, so that no additional fill is added over timing-closed blocks.

During development, top-level integration may use early release of blocks that do not contain any fill at that stage. In such situations it may be desirable to add top-level fill over those blocks for DRC checking purposes. However, while this is certainly worthwhile, the fill pattern will not be the same as if the blocks were filled locally. Therefore, it is important to obtain filled blocks at an early stage in order to uncover subtle metal density violations. This can happen because blocks are not dummy-filled to their very edge, so an empty area, a few microns wide, exists around the blocks, as illustrated in Figure 7. Trial tapeouts, conclusions

**Trial tapeouts**

Trial tapeouts allow a dummy-run to be made to identify any specific issues with stream-out, data delivery and foundry acceptance. Hierarchical designs benefit from this as much as flat ones, but issues with integration may be identified as a result of pushing towards this goal, so there are indirect benefits in addition to the obvious ones.

**Physical verification checks**

It is usual to run, as a minimum, design rule checking (DRC) and layout versus schematic (LVS) at the chip level as part of the physical verification checks. DRC ensures that the design is
manufacturable based on a runset provided by the foundry; however at the chip-level an additional runset should be used to verify the wire-bond or flip-chip connectivity requirements. This runset checks the bump/bond-pad construction and spacing requirements, although the ultimate authority on such requirements lies with the packaging house, which will also run its own checks.

By the time final tapout is reached, a list of DRC waivers should be documented and confirmed with the foundry to avoid any delays.

LVS forms the final part in a chain of verification events that should give a high degree of confidence in the functional correctness of the physical database. Throughout the physical design process, formal verification is used to check that the functionality has not changed during each step of the process, thereby forming a trail that can be traced back to the RTL code. LVS checks the physical database against the end-point of this trail and therefore confirms that the GDS correctly implements the RTL. However, there are caveats to this.

![Diagram](image)

**Figure 8** Functional and physical netlists are created in the same task so as to provide audit trail continuity.

The schematic netlist used by LVS is often different from that used by formal verification (it contains power connectivity and physical-only cells, for example). Therefore, it is important that these netlists are created by the same task, on the same database, in the same session, so that we can be confident that they are basically two different views of the same netlist (**Figure 8**). How we then use this netlist during LVS also requires some care, especially regarding the matching of top-level ports, and case sensitivity. Unless there is good reason not to, always run LVS in case-sensitive mode, and also enable options for cross-checking top-level ports between layout and schematic. Layout text objects attached to flip-chip bumps or wire-bond bond-pads will allow LVS to check connectivity right up to the pad/bump, whereas if the text is omitted, then it could be possible that unconnected bond-pads are not identified (they would just appear as floating metal, to be discarded). Clearly, any floating top-level text would also be highly suspicious.

DRC and LVS should both be run at the block level before attempting full-chip verification, but be aware that some block-level issues will only show when a full-chip verification is performed. As a rule, start running verification as early as possible, and start integrating the full-chip view as soon as
possible.

For more recent technologies (e.g. 65nm and below), we also have additional checks at our disposal. Lithography compliance checking (LCC) identifies patterns in the metal layers that are known to have an impact on yield since OPC effects etc. are liable to result in manufacturing faults or weaknesses (Figure 9). In theory, the routers used by layout tools, and the rules they follow, should be conservative enough to avoid any LCC issues, but violations are still possible and need to be understood. LCC can take many CPU-days to run, so it needs to be considered some time prior to tapeout. For hierarchical designs, it can be run at the block level for increased confidence.

![Figure 9 LCC identifies potential manufacturing faults, or weak spots.](image)

Critical area analysis (CAA) checks are also run to identify any patterns that are sensitive to particle defects in the manufacturing process. The output of CAA is an overall yield impact, since much of the analysis is statistical. Again, this can be run at the block-level to identify risk areas at an early stage.

**ESD analysis**

ESD protection is a separate subject in itself; however, it would be amiss not to mention it here. During integration, it is strongly advisable to review any ESD protection requirements for IP blocks present in the design. Often there are special power connectivity requirements, and additionally it may be necessary to add diode clamp cells and even additional IO pads to provide a suitable discharge path during an ESD event. Retro-fitting these could mean significant last-minute floorplan changes with associated schedule slippage.

**Conclusion**
This paper has only touched on some important parts of physical design for hierarchical chips. Much of what has been said is just reflecting good design practice, but sometimes these practices are not followed, corners are cut, and chips fail. If we are responsible for signing off a design, we should be asking ourselves questions like, “How confident do we feel about this chip?”, “How do these results prove our design really will work?”, and “What other checks should we have considered?”.

We cannot eliminate human error, yet human input is what differentiates one product from the next. There is always risk in building chips, but with good management and sturdy design practices we can manage the risk and head towards tapeout with confidence.

More about Steve Lloyd