IC design at advanced process nodes: Add flex to your flow

Andrew Potemski  -  August 16, 2007

To handle numerous technical challenges associated with advanced process nodes, chip designers must have a design flow that adapts to evolving requirements and design goals. At the same time, design teams must also deal with project-related challenges, such as achieving consistent design development across geographically distributed design teams, ramping up new sites, and correcting issues with third-party-library and IP (intellectual-property) quality. Today’s design flows must handle all of these challenges and still provide more flexibility, greater modularity, and new methodologies for achieving predictable design tapeouts.

To meet these goals, a design flow must be more than a means to solve design problems. It must also improve designer productivity and make design organizations more competitive. By implementing a versatile design flow and environment, design companies can:

- support multisite design efforts across geographically distributed teams with consistent sharing of design methodologies and project data;
- simultaneously handle multiple projects by quickly alerting team members to the latest best practices;
- move teams to projects with uniform flow execution;
- quickly adapt new libraries for a geometry or migrate to new geometries; and
- maximize designer productivity and minimize project delays with encapsulated methodologies and flow automation.

The Synopsys Professional Services group’s customers use many foundries and libraries, implement a range of design applications, encounter many methodology challenges, support multiple points of design handoffs, and function with distributed teams. Accordingly, its design consultants encounter a full spectrum of design- and project-related challenges. The group has created a versatile flow to handle all of these challenges.

Design flow and environment

The term "flow" has many meanings, so it is useful to clarify it and a few related terms. One of the most common flows is a reference, or reference-design, flow, which is a collection of scripts and best practices that describes the interoperability of data files across two or more EDA tools. You often tune these scripts for a targeted foundry’s technology or a target design type, such as a core.

Closely related to the reference-design flow is a design methodology—a set of techniques and best practices to implement and verify a chip. For example, a methodology to implement a 20 million-gate chip containing a crossbar switch might include hierarchical partitioning, floorplanning with a fixed-
target percentage of congestion constraint, shielding requirements on all clock and critical-signal nets, double-spacing-rule requirements for chip-level-clock and critical-signal routing, and multicorner-extraction-based signal-integrity analyses before sign-off.

In the most general sense, a design flow is a complete process that implements a design methodology or several design methodologies, such as the one previously described. You implement such a flow with a collection of ready-to-use scripts and utilities to run the tools, addressing both design- and project-related challenges. The flow might include standard practices from foundry reference flows. CAD and design teams typically configure the flow for project-specific use.

A design environment incorporates a design flow and adds the rest of the resources required to complete a chip. In addition to the flow, for example, the environment includes infrastructure, such as computing hardware and software; prescribed directory structure; data-management principles; consistent naming conventions; scripts to set up, use, and manage project data across design teams or projects; and the automation mechanics for executing the design flow and for rebuilding a chip.

**Not a pushbutton option**

Although flow automation is important for improving designer productivity, there is no such thing as a pushbutton design flow. Every design project has different requirements and input data that impose unique flow requirements. This fact argues against too much automation early in a project, when you need transparency in the processes. For example, you address many design-related challenges in floorplanning that are initially interactive, but you can automate them later in the project for rebuilding the chip after incremental changes.

Further, design flows are never static, because the underlying processes, libraries, IP, and EDA tools are dynamic. Keeping flows current requires ongoing investment and effort. A versatile, robust flow thus demands a good balance of modularity, configurability, and automation.

**Robust design environment**

You can implement a good design environment in many ways. The Synopsys group recommends starting with a set of coupled layers (Figure 1). Here, "coupled" means having layers separated by function and connected through defined interfaces.

The automation layer gives you the flexibility to configure the design flow for project-specific goals and then capture and replay selected tasks. For example, the flow-automation requirements may include flow-step selection and replay, EDA-tool-version control with support for tool substitution, and built-in error checking and reporting throughout the design cycle. "Make" files offer an efficient model for such flow-step automation, and the automatic creation of correct-by-construction make files enhances flow automation.

The project data should be well-structured with a predefined data and scripting-interface model for single-site and multisite development. Good data management enables consistent job launching and management for efficient use of hardware and software resources. For example, organizing the technology and design-related files in separate directories allows you to reuse the same technology node across multiple users and even multiple projects.

The project-data-structure requirements may include a standardized directory structure to support flow execution across multiple users and sites, variable naming and scripting conventions for reuse, and preferred data management using a freeware or commercial revision-control system for individual and group development.
Using a scalable computing infrastructure lets you accommodate the changing needs of a single project or multiple projects in parallel. Runtime and design-quality metrics are valuable for guiding the deployment and updating of computer resources, and you can extract these metrics from the design flow.

When planning a new project, program managers can develop and use project-specific metrics that take into account the design team’s size, the design’s size, memory requirements, flow-step and task durations, and even the type of available computing servers. For example, Synopsys design centers use a computer farm of Unix- and Linux-based machines, job distribution using LSF (load-sharing facility), revision control using Perforce, and software-application access using a modules file (reference 1, reference 2, and reference 3).

At the heart of the design environment, the design-flow layer incorporates the EDA-tool flow and should embed the best practices and methodologies required to address all project-design issues. The design-flow layer addresses all of the chip-implementation tasks. Design teams integrate libraries and IP from multiple sources with varying degrees of quality and completeness. Qualifying this input data early in the design cycle avoids downstream surprises.

A typical project includes libraries from multiple vendors (such as standard cells, memories, I/Os, analog- and mixed-signal functions, and hard IP) and technology files from the selected foundry. Common problems include inconsistent timing and physical views in libraries; incomplete timing, physical, and logical models for new process nodes; and missing design rules within technology files or DRC/LVS (design-rule-check/layout-versus-schematic) decks. You achieve measurable productivity improvement using a process to identify and correct these problems.

Two important processes enable designers to quickly prepare and qualify required input data at a project’s start. First, incoming quality-assurance processes should focus on evaluating libraries, constraints, and RTL (register-transfer-level) files. Second, a method to generate correct-b-construction technology files from a single, up-to-date source—often the foundry—is necessary. Note that you should separate quality-assurance methodologies from the RTL-to-GDSII (Graphics Design System II) implementation-flow steps to ensure that project teams can retarget the same implementation flow for different process technologies and multiple libraries.

You should build the functional verification upon the structured project data to allow project teams to seamlessly share design data and output between implementation and verification of the same design. With the correct setup, you can export simulation-generated data files to the implementation side without manual intervention, thereby ensuring that teams use the right file versions to analyze and implement the design.

**RTL-to-GDSII implementation flow**

An RTL-to-GDSII implementation flow should provide modular steps for use across multiple sites and projects. The flow should also allow designers to modify individual steps for design- and project-specific needs. The RTL-to-GDSII implementation flow within the Pilot Design Environment comprises five modular steps (Figure 2).

You can assemble these modular-flow steps into a design-specific hierarchical RTL-to-GDSII flow using three subflows (Figure 3). The hard_macro subflow hardens blocks, such as ARM cores and other IP for chip-level integration. The soft_macro subflow is for implementing and delivering hierarchical-design modules for chip-level integration. You use the top subflow either to hierarchically integrate the chip design with design input from the other two subflows or to provide the flexibility to simply supply a flat implementation of the chip design.
This subflow architecture allows each project team to structure, configure, and automate the flow for its block-specific needs. Further, it gives project teams the flexibility and control to apply the same implementation flow to a range of design applications and design styles and to ensure consistent integration across multiple sites and handoffs.

Another key benefit of this approach is the ease with which users can introduce advanced EDA-product methodologies, reference flows, and best practices without rebuilding the baseline flow from scratch.

This flow implementation supports a variety of design methodologies and tool features to address design issues common in complex SOCs (systems on chips), such as high frequency, power management, signal integrity, reliability, yield, and testability. For recommendations on specific design methodologies for design planning, hierarchical planning, power management, and signal integrity, see Reference 4.

**A measured approach**

An important consideration when building a design flow is the means for collecting metrics associated with project execution. The monitoring and reporting of metrics throughout the design cycle can be of great value in helping designers assess design-completion status.

Some view with skepticism the need for determining relevant metrics and establishing techniques and discipline for measurement regarding these metrics’ accuracy and value. Imperfect as metrics may be, measuring the performance of the design flow in real time is an essential element to improving productivity. Flow metrics help designers and program managers determine where extra resources and time are necessary to ensure that teams complete design tasks on time.

Also, monitoring key metrics improves predictability by providing visibility into the health of the design and tapeout readiness. In the longer term, flow metrics help identify aspects of the flow that require improvement and provide a basis for quantifying the improvement from one flow version or project to the next. Additionally, metric reports improve communication of design-project status among stakeholders, and teams can maintain reports on a project-tracking Web site or a project scoreboard for postproject analysis.

To avoid distraction from primary design tasks, capturing and reporting design metrics require automation for minimal overhead. Automated metrics simplify evaluation of tool-change needs, assessment of new-methodology choices, and identification of flow bottlenecks. You can accomplish automation by embedding the metrics-tracking system within the flow architecture (Figure 4).

Sample metrics include the design’s physical size, cell use, number of nets, number of pins, number of instances, and the physical DRC status. Detailed QOR (quality-of-results) metrics relating to timing closure include the TNS (total negative slack), WNS (worst negative slack), number of timing violations, slack histograms, number of clocks, clock skew, and clock latency.

Equally important are the metrics associated with infrastructure resources. The metrics-monitoring facilities in the design flow can capture detailed statistics on CPU runtime, memory usage, and disk-space usage, as well as on EDA tools and versions. Reference 4 contains a more detailed discussion of how to use metrics.
Lessons learned

Design-flow development is not an optional activity, though the efficiency and impact on design programs can vary greatly. Complex chips require a robust but flexible system to meet schedules and ensure tapeout success. Shorter market windows have put tremendous pressures on designers to meet aggressive schedules and performance targets. Without a well-defined flow and environment, designers have little chance of meeting these aggressive goals. Actively maintaining up-to-date design flows can help remove flow issues from the critical path. A well-defined environment also gives designers on-the-fly configurability to address new issues as they arise.

Second, attempting to create a one-size-fits-all flow is an unreasonable approach. Such an approach is expensive and is unlikely to support all the methods that CAD engineers and designers need. A practical approach is to create one modular, layer-based design environment in which you configure the design flow for project- and design-specific needs. The result is an easy-to-set-up, easy-to-use, and easy-to-support flow.

For example, a design can easily accommodate library and IP changes if you separate the technical-library preparation and library-qualification methodologies from the RTL-to-GDSII flow steps, allowing designers to target the same implementation flow for multiple processes, technologies, and libraries. Similarly, a team can adopt new EDA tools and features if the flow steps are modular and separate from the automation layer.

Third, to maximize designers’ flexibility, you should automate the design tasks and not the process. The focus should be on capturing design-specific tasks and ensuring repeatability of the tasks through automation. You can accomplish this task by separating the baseline flow from the automation and the project-data-structure layers.

Additionally, to ensure predictable implementation across multiple sites and multiple projects, apply a logical and consistent project-data structure, but do not hard-code the data structure in the design flow. Use variables within the design flow and common directory-naming conventions for the design tasks.

No matter how robust the design flow, nothing can replace the expertise that comes with experience and proper training on new tools and methodologies. This requisite knowledge helps project teams avoid what are often wasted iterations through the design process.

Finally, the development of the flow and environment is not a one-time activity, because processes, libraries, IP, EDA tools, and—more important—the underlying designs change constantly. Without periodic methodology enhancements, tool updates, and ongoing support, it is easy for the flow to become outdated, resulting in lost designer productivity and project delays.

The benefits of setting up a versatile design environment are numerous for both the CAD and the project teams but boil down to a simple concept: To achieve predictable time to results, you need a predictable design process. Furthermore, by making the design processes consistent and measurable, you can make systematic improvements, quantify results, and more easily share results among design teams. A robust and versatile design environment demonstrably and dramatically improves designer productivity and tapeout predictability, and you can deploy it with minimal overhead.

References
Author's biography

Director of the Design Flow Center of Excellence at Synopsys, Andy Potemski has more than 25 years of experience in the semiconductor and EDA industries. He has worked in design-engineering, consulting, and management roles at Synopsys and at IBM. Potemski graduated from Fairfield University (Fairfield, CT) with a bachelor’s degree in electrical engineering and holds four US patents in IC design.