TECHNICAL **BRIEF**

cadence®

# Computational Software: A New Paradigm for EDA Tools

EDA tools have been evolving since the mid-1980s. The development can be broken down into three major phases, and it's important to understand these three phases to realize where EDA tools are now, where the tools are much more tightly integrated, and where they are starting to employ machine learning techniques to deal with the massive level of complexity.

This evolution has resulted in three key innovations of modern (late-stage Phase Three) computational software, including integration of previously independent design tools, partitioning to multiple CPUs and servers, and using machine learning to improve productivity.

## Phase One

In the beginning, companies were started to develop, maintain, and evolve state-of-the-art algorithms: the best routing algorithm, the best timing algorithms, the best synthesis algorithms, etc. Often companies were created to develop one specific tool with experts in that domain. Semiconductor companies had large CAD groups with hundreds of people who would take these tools and develop flows around them with scripting languages so they could actually create a chip.

This era was known as "best-in-class point tools." A second part of Phase One was when companies like Cadence acquired some of these companies to create a broader product line. For example, Cadence acquired Tangent for gate-array and cell-based place and route (P&R), Gateway for Verilog simulation, and Valid for printed circuit board (PCB) design. Cadence had already developed (going back to SDA and ECAD days) the Virtuoso® environment for custom and analog layout and Dracula for DRS.

At the end of Phase One, the EDA industry consisted of a number of companies like Cadence with fairly broad product lines (including some major holes), plus a lot of single-product startups.

## Phase Two

Phase Two started when the EDA companies recognized that state-of-the-art algorithms had to start working better together. For example, timing-driven P&R requires a timing engine to be part of the tool. The semiconductor CAD groups were challenged when the timing engine in the P&R tool was different from the timing engine in the simulator or, once it came along, static timing analyzer. The tools were not integrated, and it was unsatisfactory if the P&R tool made the engineers think they had met timing, but the timing analysis tool did not. With many tools, particularly synthesis, engineers had to overcon-strain the problem as if they were negotiating with the tool. Engineers would ask for 600MHz and hope to get 500MHz.

The solution was to have common engines for common functions so there would only be one answer to any given question. This was a massive undertaking, akin to changing the oil in a car without stopping. Changing the timing engine in a synthesis tool without breaking it is not a simple undertaking.

By the end of Phase Two, most EDA tools had shared placement, timing, extraction, design-rule checking, and other engines.

## Phase Three

We entered Phase Three when processor speeds capped out and processor companies delivered increasing power through multi-core. Some EDA algorithms could take advantage of this fairly easily (many DRC rules can be checked independently, for example). But many struggled, notably simulation with a global concept of time and causality. Large semiconductor and system companies put in place server farms with tens of thousands and then hundreds of thousands of processors. EDA tools had been created in an era where the main way to get more compute power was to wait for a faster processor. Now, with big data centers and multi-core processors, there were vast amounts of compute power available but in a way that the design tools could not utilize. Phase Three would emphasize the repartitioning of these tools to leverage modern computing fabrics.

Traditionally, the EDA process was split into different steps: synthesis, placement, routing, extraction, timing, and signoff. With the common engines developed in Phase Two, these worked better than when all the engines were different in Phase One. But the new, better way is to partition the design, as much as possible, so all the phases interface tightly on common data structures but don't handle the whole design. When that works, a big design can be scaled to many cores/servers. Some algorithms still are really hard to parallelize, such as placement (which is inherently global, at least at the start of the process). But other algorithms are much more straightforward. For example, detailed routing on one part of a chip doesn't really interact with routing on another part of the chip once the global router has divided up the task.

Even interactive programs benefit from this tighter integration, being able to open an editing window on a chip from inside the package editor, or vice versa, without having to explicitly read and write files and switch tools.

Another part of Phase Three has been the addition of deep learning approaches to guide the algorithm selection under the hood. A lot of what engineers do when running EDA tools, towards the end of the design, is look at the results from one run, tweak a few parameters, and then do another run. Deep learning allows the tool to tweak the parameters itself in an intelligent way. In this case, intelligent means learning from other similar designs with the same design style at the same company. It also includes learning from earlier runs of the tools on the same design.

## Computational Software

That brings us to the three key innovations of modern (late-stage Phase Three) computational software:

- The integration and co-mingling of previously independent design, analysis, and implementation to achieve optimal results, which delivers optimized SoC/systems that are achieved more automatically with a more predictable path to closure.
- The partitioning and scaling of computation to thousands of CPU cores and servers, which deliver orders-of-magnitude faster computation for exploring more alternatives for more optimal results and leveraging more abundant compute cycles.
- The introduction of machine learning to improve and harness design heuristics for system optimization, which ups the level of abstraction for human user experience and control, automates previously burdensome routine tasks, and finds more optimal results in a sea of too many alternatives for humans to consider completely.

## Summary

Throughout the evolution from point tools geared to solve one specific problem in the mid-1980s to today's very integrated design tool flows, there's been a lot of innovation. Designers have benefited, and will continue to benefit, from tighter integration between various tools to more quickly get the results they need for complex SoC designs. As these designs have become bigger and bigger, the tools have been redesigned to take maximum advantage of multiple CPUs and servers. Finally, now machine learning is helping find more optimal design results.

Cadence was at the forefront of these trends and continues to push the envelope to develop better and faster tools for its customers. The next ten years should be even more exciting!

cadence®

Cadence is a pivotal leader in electronic design and computational expertise, using its Intelligent System Design strategy to turn design concepts into reality. Cadence customers are the world's most creative and innovative companies, delivering extraordinary electronic products from chips to boards to systems for the most dynamic market applications. **http://www.cadence.com.**