TODAY, the practical reality of functional design verification is that designers depend on a portfolio of techniques to achieve acceptable levels of confidence in the correctness of their designs. Various verification techniques such as functional simulation, emulation, and formal verification (in its various flavors) make up this portfolio. Each technique comes with strengths and limitations, making all of them relevant but not complete for the overall verification task. Designers and verification engineers are becoming increasingly better at understanding the niche areas in which each technique excels. This understanding of the niche areas helps the design community deploy the portfolio for verifying complex modern designs. It also leads to a better understanding of the interactions between the techniques.

This greater understanding has enabled design and verification engineers to exploit the synergies, or beneficial interactions, between the various verification techniques and also among other parts of the design flow. This special issue focuses precisely on these emerging synergies. Before delving into how the selected articles of this special issue highlight these synergies, it is instructive to examine the verification portfolio and the historical development of its various components.

Functional simulation

The earliest member of this portfolio, which continues to be the workhorse, is functional simulation. The history of functional simulation probably goes back to the use of computers in digital design itself, when early schematic capture tools provided the ability to run simulation vectors on the design’s schematic model. The major breakthrough in functional simulation, however, came with the development of hardware description languages such as Verilog and VHDL, and the accompanying simulators in the late 1980s. This breakthrough immediately provided scalability in terms of describing large designs; schematic entry limited design size. However, being able to describe large designs is not sufficient; increasing design sizes rapidly increases the space of stimuli that you must validate for these designs. A design with $k$ inputs has $2^k$ possible input vectors. The fact that you might need to simulate a long sequence of input vectors in a particular simulation run further exacerbates this problem. The length of the sequence somewhat relates to the number of possible states the design can reach, which is $2^n$ for a design with $n$ state elements. This exponential growth of simulation requirements results in significant simulation times, even for what is necessarily only a very small coverage of the space of possibilities that you must test. The use of the Verilog and VHDL languages led to an interesting synergy between verification and design itself, when designers used design specifications in these languages not just for simulation, but also for design via automated synthesis.

Emulation

Emulation made its appearance shortly after that: Instead of simulating a software model of the design, you could run the stimuli on a hardware prototype of the design. The simultaneous development of FPGA technology, which permitted the programming of any arbitrary logic design onto such programmable logic, enabled emulation. Again, as with simulation, there is an interesting synergy between emulation and design synthesis. Essentially, designers could take the synthesis algorithms for designing digital circuits and modify them for mapping the designs to the underlying FPGA fabrics. Although emulation systems provide two to four orders of magnitude speedup over software simulation,
they come at a cost. Differences in the FPGA’s size and that of the designs being validated cause the need for boards of FPGAs to program cutting-edge designs. Further, partitioning the design onto this set of FPGAs is a complex task. The net consequence is that, although emulation provides significant speed benefits, it ends up being significantly costly in its implementation. Designers thus reserve emulation for high-cost (that is, high-volume) projects. Even with the speed benefits, emulation retains the important limiting characteristic of simulation so that it can only validate a tiny fraction of the possible input stimuli.

Formal verification

The need to overcome this limitation in emulation motivated research and subsequent development in formal verification. Formal verification’s basic premise is that a mathematical proof of correctness implicitly covers all possible stimuli, thus avoiding the explicit enumeration of an exponential number of stimuli that simulation and emulation require. Although this motivation is unassailable, translating it to practical reality has been a challenge. Nonetheless, significant research and development efforts in this area have yielded practical success along two distinct directions.

Along the first direction, the semiconductor industry has seen significant success in equivalence checking, an area of formal verification. Correctness is defined as the equivalence of two designs—which is certainly not the same as proving that a design has a desirable property. Fortunately, equivalence checking can be localized if the designs have corresponding latches. Once the correspondence between latches of a reference design and an implementation has been discovered, equivalence is just a question of showing that corresponding latches have the same next-state function. This has proved very valuable in validating that an implemented gate-level design matches its desired behavior as specified at the RTL. In fact, synthesis enabled this specific application of Boolean equivalence checking. The realization that synthesizable Verilog could help define hardware models for equivalence checking at both the RTL and gate levels was a tacit, but also very important, synergy.

Along another direction, the industry has seen development in the broad area of property checking. Property checking is also referred to as model checking, which is the underlying technology enabling it. The focus of this direction is to check whether the design, either in its specification or its implementation, satisfies certain properties. Unlike equivalence checking, which limits itself to one step in time, property checking considers all time into the future. For example, checking that a design never deadlocks, or that each request is eventually responded to, requires the consideration of sequences having an unbounded length. In contrast to equivalence checking, which is well accepted (albeit in the limited domain of checking gate-level implementations against RTL specifications), property checking does not clearly own the verification turf in any part of the design flow. Instead, it is carving its niche in establishing the correctness of interfaces between components, as well as in uncovering difficult-to-find buggy corner cases in designs. Like equivalence checking, model checking took advantage of the fact that it’s possible to use synthesizable Verilog (or VHDL) to define hardware models for use in model checking. There are also well-known synergies between some of these verification technologies. For example, equivalence checking and property checking, although very different fundamental techniques, use the same or similar reasoning mechanisms such as Boolean satisfiability (SAT), and function representations such as binary decision diagrams (BDDs). Similarly, the basic idea of proofs over all inputs in formal verification has been married with functional simulation to give rise to what is called symbolic simulation. Another synergy between formal verification and functional simulation is evident in the use of assertion- and constraint-based verification. The use of assertions started in the formal verification community as a mechanism to specify properties (especially temporal properties). This idea migrated to the simulation community with the standardization of assertion libraries and assertion languages. Designers now use constraints to define the environment of a unit under verification and can even be used to generate random simulations.

In this issue

The theme of synergies between various verification methodologies, and between design and verification, continues to be alive in the verification efforts today as exemplified by the four articles selected for this special issue. Each of these articles either amplifies a known synergy through an interesting case study, or exposes a novel one by combining technologies in a useful way. All of the articles illustrate their methods on actual designs.

Zambaldi et al. of Infineon present an article about testbench technology that allows the reuse of models for different purposes—if the testbench has been programmed in a modular way. By simply changing adapters—that is, interfaces between parts of the test-
bench—the same testbench is usable for high-level verification, RTL verification, and emulation, and it can even run on a tester. One interesting consequence of this construction is that the entire testbench can run on an emulator—at emulation speed! Of course, to accomplish this, the testbench must be synthesizable.

In the second article, former Hewlett-Packard researchers Tasiran, Yu, and Batson investigate an altogether different type of synergy. The authors use a high-level specification and formal verification (model checking) to specify a design. They then use the same model checker to monitor simulation to see whether simulation follows the high-level design model. They accomplish this by mapping the implementation in RTL into the high-level model, using a unique two-step process in which architects provide half of the mapping and design engineers provide the other half. The authors illustrate their technique ex post facto on an HP design.

Next, the article by Kim and Kyung examines synergies between functional simulation and emulation. Specifically, it considers the scenario that validates part of the design using simulation and another part using emulation. Kim and Kyung argue that communication costs among the verification system’s hardware and software parts must drive the partitioning of the design into simulatable and emulatable parts. However, the use of such partitions could require synthesizing testbenches, which might not always be possible with behavioral testbenches. The authors show how to overcome this limitation by providing a method to synthesize behavioral testbenches.

The fourth article presents a synergy in a different direction. Bhadra, Krishnamurthy, and Abadir of Motorola’s PowerPC group investigate the integration of two equivalence flows, one for test and the other for functional equivalence. They observe that proving the equivalence in each case to the switch-level model requires two slightly different sets of design constraints. A careful examination of the constraints necessary for the different equivalence problems can reveal a common set of constraints for each switch-level model. Rectifying the two constraint sets saves re-spins and debugging in silicon.

Finally, the general interest article by Sheng and Hsiao is a synergy of underlying algorithms. The article describes how to compute inverse images—used in model checking—with a unique hybrid technology using state-of-the-art ATPG combined with a binary decision diagram approach. The authors provide experimental results to validate that their new algorithm is more efficient than existing techniques.

**The Holy Grail** of all efforts in verification is to determine a comprehensive verification methodology capable of verifying arbitrary designs. This is still elusive. For now, most efforts are content with developing and improving specific techniques, each of which is relevant in some niche area of verification. However, as we have seen, in the process of improving these techniques there is a tremendous cross-pollination of ideas between these areas and with the traditional design flow, leading to greater strength in our efforts to tame the verification problem. Perhaps it will be through these very synergies that we will achieve our elusive goal of a comprehensive verification methodology.

---

**Carl Pixley** is senior director of the Advanced Technology Group at Synopsys. His research interests include formal verification, constraint-based verification, and transaction-level to RTL-level verification. Pixley has a BS, an MS, and a PhD in mathematics from the University of Omaha, Rutgers University, and SUNY-Binghamton University, respectively. His scientific and engineering inventions include the Pixley-Roy topology, the Eaton-Pixley-Edwards-Miller theorem, and the binary-decision-diagram-based approach to model checking, sequential equivalence, resetability, and constraint-based verification. He is a member of the IEEE and the Mathematics Association of America.

**Sharad Malik** is a professor in the Department of Electrical Engineering at Princeton University. His research interests span all aspects of EDA, with a focus on the synthesis and verification of digital and embedded computer systems. Malik has a BTech in electrical engineering from the Indian Institute of Technology, New Delhi, India; and an MS and a PhD, both in computer science, from the University of California, Berkeley. He is a Fellow of the IEEE, general chair of the 2004 Design Automation Conference, and associate director of the MARCO/DARPA Gigascale Systems Research Center.

Direct questions and comments about this special issue to Carl Pixley, Synopsys Technology Park, 2025 NW Cornelius Pass Road, Hillsboro, OR 97124; cpixley@synopsys.com.