Special Technology Report

Mixing Formal and Dynamic Verification, Part 1

By Bill Murray

Over the last few years, there has been a noticeable uptick in the use of formal verification to augment dynamic verification. Given that both techniques leverage assertions [1, 2], one would assume that there would be a great deal of collaboration between dynamic testbenches and formal property checking, the user teams and the tools. Indeed, a DVCon 2009 panel discussed mixing the two – thus the title of this special technology report (STR).

In this STR, we dig down to the use case level to determine how formal is being used, and how it augments dynamic verification. We used 17 use cases ranging from early RTL analysis to functional sign-off to survey 19 engineers and engineering managers at 16 industry-leading IP, chip and systems design companies to understand their formal adoption and use in 2006 and 2009, and projected use in 2012. We also interviewed respondents from 9 of those companies to gain even more detailed insight. Of course, the user responses apply only to their individual groups, not necessarily throughout their multi-divisional companies.

We found that there is certainly collaboration between simulation and formal verification, but "mixing" might be a bit of an overstatement at this stage. The methodologies, standard common coverage metrics, and tool interoperability required by true mixing are in a distinctly embryonic stage of development.

The good news is that formal verification has developed to a level of usefulness and maturity at which design and verification teams *want* to mix it with tried and more-or-less trusted dynamic verification. What has changed in formal verification to make this possible?

In part 1 of this STR, we:

- Discuss the upper-level results of our use case survey.
- Learn from the respondents why formal is enjoying increased adoption.
- Review formal's sweet 'n' sour spots and how the sweet spots are expanding.

In part 2 of the STR, we discuss:

- The detailed results for the 17 use cases.
- How formal is currently being used with dynamic verification.
- The application of formal in the ESL space.
- How technology users and providers envisage formal methods in 2012.

Now let's review what users at Alcatel-Lucent, Analog Devices, ARM, Cisco, DE Shaw Research (DESRES), Fujitsu Microelectronics Europe, HP, IBM, Infineon, Intel, nVidia, Qualcomm, Saab, Silicon Logic Engineering (Tundra), STMicroelectronics and Sun Microsystems said to SCDsource. This is their story.

SCDsource User Survey

User Base Broadened

Of the 19 respondents, 14 were employing formal verification in 2006 – this STR refers to them as established users. The five respondents who adopted formal after 2006 (dubbed "recent adopters") constitute a 36 percent increase in user breadth from 2006 to 2009.

Use Employment Deepened

Our survey shows a significant deepening of proliferation within user groups. Figure 1 shows the average number of use cases employed in 2006 and 2009, and projected employment in 2012. Average use cases grew 46 percent from 6.6 in 2006 to 9.7 in 2009. Our respondents project this to grow by 32 percent to 12.8 use cases in 2012.

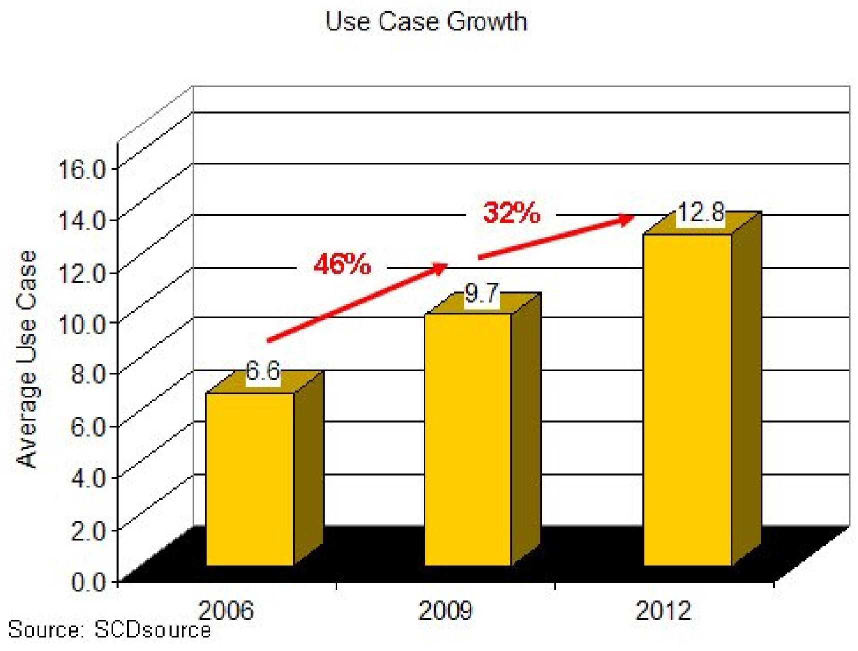


Figure 1: The deepening use of formal verification (Source: SCDsource)

Total Use Nearly Doubled

So, what is the total formal deployment across the sample of 19 user groups? The user breadth and use depth are combined in figure 2. In this population, the use of formal in 2009 is nearly double that in 2006, and 2012 use is projected to increase another 32 percent over 2009. Projected 2012 use is about 2.6X that of 2006. This equates to a compound annual growth rate (CAGR) from 2006 to 2012 of over 17 percent.

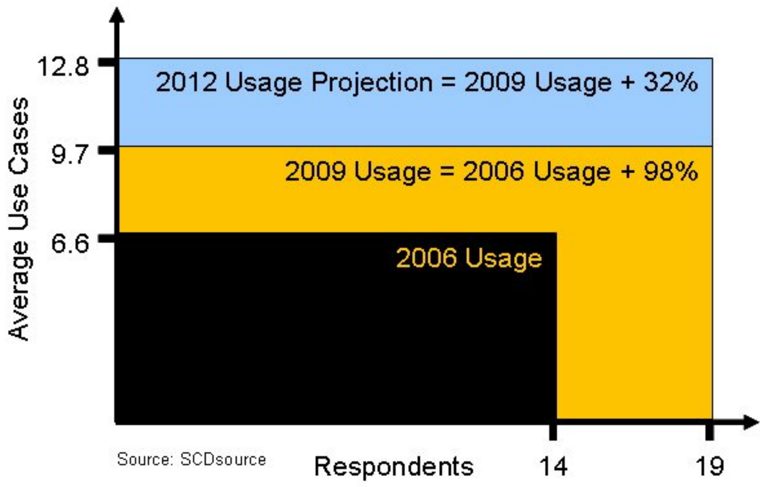


Figure 2: Formal verification use nearly doubled from 2006 to 2009 (Source: SCDsource)

In summary, one could argue that the growth rate in this sample is quite impressive for a verification approach that has a reputation for being difficult to adopt and use.

Recent Adopters Ramp Fast

How difficult to adopt and use? Our survey shows that recent adopters are setting an aggressive pace. Could this mean that formal is becoming easier?

The 5 recent adopters grew their average use case employment from zero in 2006 to 7.8 in 2009, and project that this will grow by roughly 39 percent to 10.8 in 2012 (see figure 3).

The average use case employment by the 14 established users in 2006 was 6.6, and grew about 56 percent to 10.4 by 2009. This group projects that it will expand employment to 13.5 by 2012, a growth of about 30 percent.

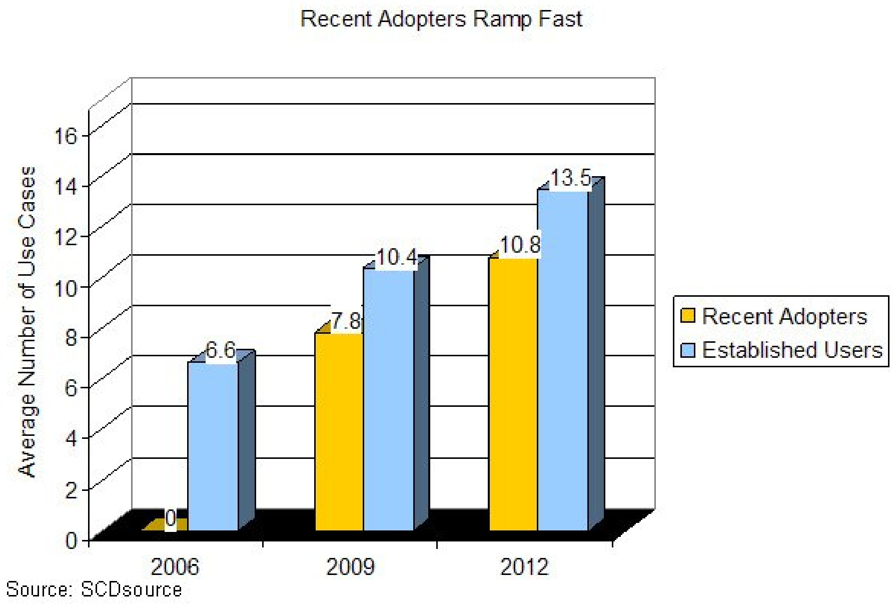


Figure 3: Recent adopters are catching up – fast (Source: SCDsource)

The recent adopters are employing more use cases in 2009 (7.8) than the established users were employing in 2006 (6.6). If use deepens according to projections, the recent adopters' use case employment in 2012 (10.8) will be slightly higher than that of the established users in 2009 (10.4). Recent adopters in this population will thus lag established users by about three years – but bear in mind that many established users adopted formal quite a long time ago. Clearly, recent adopters are catching up with established users – fast.

Advancing Backwards in the Flow

Formal is renowned for its ability to perform exhaustive verification – and for the decades-long perception that the user must have a Ph.D. in maths to undertake it. However, formal's proponents claim that modern tools and methodologies make it accessible to mainstream verification engineers. Indeed, they claim that formal is expanding into the design domain. Our survey shows that they are right on both counts.

The use case expansion – shown in much more detail in part 2 of this STR – clearly shows formal's advance into mainstream verification, but what about design? One example of formal's expansion into the design domain is that of early RTL analysis. It is already among the most widely-employed use cases in 2009, used by 68 percent of our respondents. Moreover, our survey projects use to increase by the year 2012 to 95 percent of respondents (see figure 4).

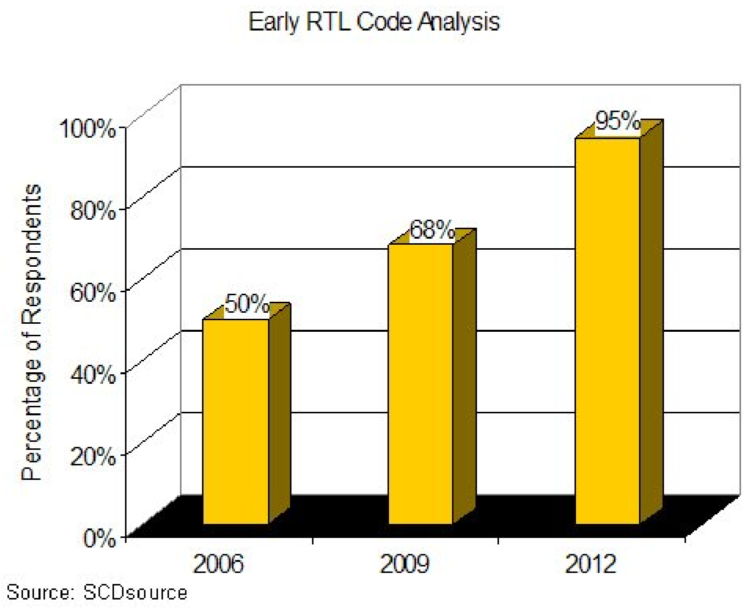


Figure 4: Formal has advanced backward into design (Source: SCDSource)

Formal has expanded its scope, but that doesn't mean that it has lost its focus. Its major traditional perceived strength – exhaustive verification – is being performed by 68 percent of respondents in 2009, expected to increase to 84 percent by 2012 (see our **cartoon** at the end of part 1). Exhaustive verification of block features, functions, operations, and transactions remains the leading use case in terms of importance, with about 53 percent of respondents ranking it in their top 3 use cases.

Clearly, formal is fulfilling its original exhaustive verification mission, but equally clearly, it is has advanced backwards in the design flow – and into mainstream design and verification.

Tools Revenue Numbers

According to Gary Smith at Gary Smith EDA, the CAGR for formal tools revenue over the period 2007 to 2012 is about three times that of simulation tools. Specifically, formal verification and formal analysis revenues are growing at a CAGR of 4.2 percent and 4.3 percent respectively, while RTL simulation is growing at only 1.5 percent.

The absolute revenue growth results for formal appear to conflict with a common perception – and with our survey results – that it is enjoying noticeably greater proliferation than heretofore. According to Smith, these modest revenue growth numbers are the result of competitive price pressure in RTL design and verification tools in general, especially from Cadence, Mentor and Synopsys.

Formal Adoption - Why Now?

For decades, proponents of formal property checking have claimed that it can achieve higher verification quality because it can reach parts of the design that simulation cannot reach – at least, not efficiently in a bounded project time. They claimed not only that formal can verify that a design behaves according to the specification, but also that its exhaustive nature enables it to detect unintended behavior far more effectively than can dynamic verification. Early in the life of formal property checking, some claimed that it would one day replace simulation. It hasn't. In fact, only in the last few years have we seen it proliferate beyond a relatively small community of formal verification experts into the mainstream. Why? What is happening to fuel this long-predicted and long-awaited move? We start with the crumbling barriers to adoption.

Crumbling Adoption Barriers

Historically, formal verification has been perceived to be capacity-limited, difficult to use, and lacking robust, systematic methodologies. These barriers have been gradually crumbling.

Capacity

According to Jason Baumgartner, formal technology lead in IBM's Systems and Technology Group, "For decades, formal algorithms bordered on the 'unusably unscalable.' The user had to target a piece of the design small enough to avoid choking the formal verification tool, requiring copious expertise and effort."

IBM has significantly increased scalability by employing advanced techniques such as algorithms that automatically reduce the size of the problem to be analyzed, and which alternate effort between 'proving" and "disproving." Baumgartner said "We can now tackle the verification of large blocks such as an entire floating point unit (FPU). This increased capacity eliminates, for example, the need to whittle out from the FPU some small state machine that handles some complex bypassing, the standalone verification of which would otherwise require orders of magnitude greater manual effort."

Most respondents agree that capacity has improved, but many take the same view as Alcatel-Lucent's Joachim Knäblein. He is in an ASIC design support team that works on verification methodology and internal tools, and wants the capacity of formal tools to increase further.

Ease of Use

Saab Avitronics started investigating formal methods and commercial tools as far back as 1996, according to Håkan Forsberg, technical specialist. "As recently as 2004 to 2006, dynamic (simulation) approaches were still better at verifying our designs – which are largely control-centric – despite improvements in the formal tools. We had to use a lot of constraints to make the formal tools work," he said. "Now, it's easier to write assertions and constraints, but it's not necessarily more automated."

The historical lack of industry-wide standard assertion languages created a barrier to formal adoption. Tim Stremcha, a senior design engineer at Silicon Logic Engineering (SLE), observed that "Formal verification became more compelling with the advent of standard assertion languages for both formal and dynamic verification. A standard language enables us to import formal assertions into the dynamic testbench environment." Stremcha finds this especially helpful while the testbench is still under construction. He said "Mixing formal assertions with the testbench allows us to co-develop the two verification approaches. In effect, importing a new assertion provides a new directed test when running formal, and the input constraints provide a very nice cross-check of the testbench stimulus."

According to Volkan Esen, an ABV specialist in Infineon's Enabling Technologies and Services (ETS) group, the popularity of assertion-based simulation has spurred the adoption of formal verification. He said "ABV knocked down the notation barrier to formal adoption. Users have stepped out of their VHDL comfort zone to write assertions in PSL and SVA, and discovered that it's as intuitive as a waveform spec. These languages eliminate the hardcore maths."

Methodology

Methodology is key. Summing up the view of many respondents, Bryan Dickman, Director, Design Assurance at ARM, said "the performance and capacity of formal tools, though important, are less important than having the appropriate methodologies to verify a particular class of problem. And you still need engineers experienced in formally verifying that class of problem. We spend a lot of time developing methodology with our tool suppliers and are seeing good progress toward some consistent use models."

Of course, improved capacity, ease of use, and methodologies reduce barriers to adoption, and all undergo a "continuous improvement" process. But what are the incentives for using formal verification at all?

Quest for Quality

According to our interviewees, it's primarily about verification quality. Productivity is a key factor in the return on investment (ROI) calculation that often determines whether formal is deployed in any given situation. So, higher productivity is a hot button requirement in formal deployment, if not always a driving factor.

Safety First

According to Saab's Forsberg, "We must comply with an avionics industry verification requirement specified in the Radio Technical Commission for Aeronautics (RTCA) document DO-254 – Design Assurance Guidance for Airborne Electronic Hardware. These guidelines [3] mandate the use of additional design assurance methods to verify complex safety-critical designs at the two highest design assurance levels. Proposed advanced verification methods include safety-specific analysis, elemental analysis, and formal methods."

According to consultant Kristoffer Karlsson, who has worked with both Saab and the European Aeronautic Defence and Space (EADS) company, formal methods afford the design team a more effective means to:

- Analyze design intent and its implications for functional verification.
- Validate functional requirements within their operational context, and
- Validate the correctness and completeness of the assertions, whether used in formal or dynamic assertion-based verification (ABV).

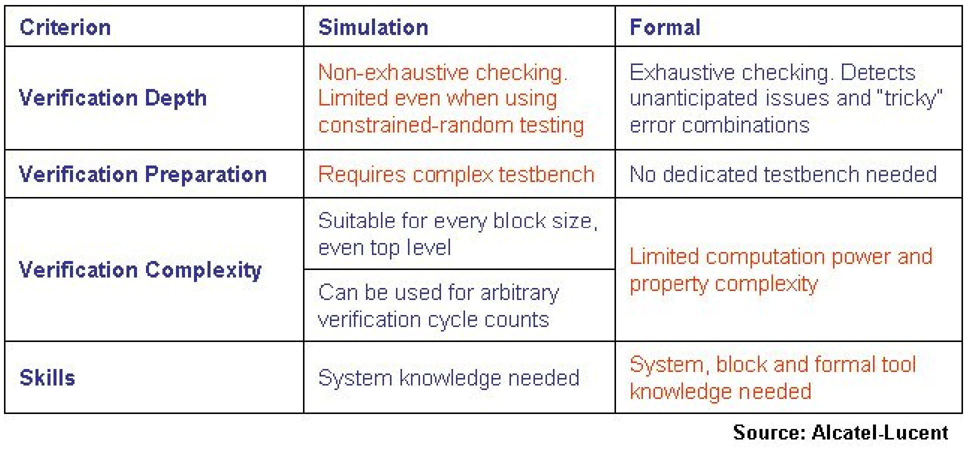
Bug Hunting First, but Sign-Off is the Ultimate Goal

We asked a recent adopter why his group adopted formal methods. Hewlett-Packard's David Lacey is the verification manager for chipset designs targeted at the company's high end servers. Initially, Lacey is tasking formal with bug hunting, but he envisages it as a sign-off criterion over the longer haul.

"Our goal is to get high-quality silicon out in as few releases as possible," said Lacey. "There are certain classes of bugs that are very difficult, if not impossible, to find with simulation, and so that's where we'd like to derive value from this technology. Just in general, it's another way to uncover bugs that we might otherwise miss."

Exhaustively Verify – No Anticipation Necessary

Alcatel-Lucent's Knäblein tabulated a comparison between simulation and formal verification (see figure 5).



Criterion	Simulation	Formal
Verification Depth	Non-exhaustive checking. Limited even when using constrained-random testing	Exhaustive checking. Detects unanticipated issues and "tricky" error combinations
Verification Preparation	Requires complex testbench	No dedicated testbench needed
Verification Complexity	Suitable for every block size, even top level	Limited computation power and property complexity
	Can be used for arbitrary verification cycle counts	
Skills	System knowledge needed	System, block and formal tool knowledge needed

Source: Alcatel-Lucent

Figure 5: Comparative benefits and costs of simulation and formal methods (Source: Alcatel-Lucent)

He cited verification quality as the main driver for formal adoption. "We ran simulation and formal benchmarks on an ASIC design. We found about the same number of bugs in each, but not the same bugs. Simulation found the more obvious bugs – those which occur at longer system run times. Formal found the more exotic bugs, bugs which seldom occur, but which can be quite serious."

Knäblein pointed out that "Simulation can make sure that something works the way it's supposed to work, but it cannot necessarily ensure that something does not work the way it's not supposed to work. For example, formal found a register map bug whereby some registers were mirrored in the address space. Accessing this address evoked an unexpected response from the register map. Normally, our simulation just checks whether the registers are accessible; it doesn't check whether invalid addresses do not respond to an access."

Infineon has used formal verification for nearly a decade, according to Darren Galpin, a verification team lead. Galpin's group verifies silicon IP such as bus bridges and data mover engines [4] for the company's TriCore single-core 32-bit MCU-DSP architecture. "We use formal to verify interface blocks that translate from one communication protocol to another. It's not necessarily terribly complex, but it is critical that we get it right," he said. "With an exhaustive formal proof at the operation/transaction level, we can be sure that we've completely covered a block's functionality. Using random and/or directed tests would require us to generate rather large coverage sets – and we still couldn't be sure that we've covered everything." Galpin observed that formal is particularly good at detecting bugs in legacy IP.

Exhaustive formal verification is especially useful in SoC design, where a great deal of functionality is implemented in software, according to Wolfgang Ecker, principal engineer in Infineon's ETS group. He observed that "Formal eliminates the need to generate individual, targeted test cases and checkers. For example, when we verify the correct integration of a module or IP into the SoC, we must verify its correct connection and communication with the system bus. Using a non-formal approach, we would first have to generate the transaction or bus access on the CPU side – essentially a software instruction. We would then have to run several checks to ensure that all relevant registers are modified, and that all other registers remain undisturbed. In contrast, exhaustive formal verification executes a complete search of the entire function space, driven by specifications generated from IP-XACT with SPRINT extensions. We don't need to worry about software accesses and data bursts."

Advancing Backwards in the Design Flow

Olivier Haller, verification methodology manager at STMicroelectronics, said "We've been using formal verification for about ten years. We adopted a mixed formal and dynamic ABV methodology on blocks and IP about a year ago." Haller is responsible for the definition, deployment and promotion of innovative verification methods across all ST divisions. He states that ST's block and IP designers use assertions to:

- Perform some initial verification.
- Capture design assumptions which assist the integration of the block into chip-level verification.
- Ease and speed debug because assertions are often closer to the root cause of failure than dynamic black-box verification data.
- Communicate desired verification coverage points to the verification engineer.

Richard Ho, researcher at DE Shaw Research (DESRES), said "We use formal methods to complement simulation and have achieved improved quality of results (QoR). For instance, when a coverage point is missed in simulation, we use formal to determine whether or not the point is reachable. If it is, a simulation test case is constructed to find it, based on the formal verification counterexample."

Ho continued "Also, early RTL code analysis using very simple, automatically-generated assertions is invaluable in achieving coverage closure. Formal helps us to rapidly identify sections of code that are complex and may cause coverage closure challenges for simulation. There is a close correlation between these complex sections and the verification challenges that we encounter further down the flow, so their early detection allows us to modify RTL code at the point in the flow when it is easiest to modify." [5]

Silicon Logic Engineering's Stremcha works at the designer-verifier interface. "We use formal at the design stage to improve the quality of design – independent sign-off verification by the verification team remains mandatory." The company develops silicon intellectual property in support of its ASIC and FPGA design services. According to Stremcha, "Configurable IP, such as our chip-to-chip interface protocol core that supports a range of SERDES lane counts and speeds, is a very good candidate for formal property checking. Formal enables the designer to verify a set of configurations supported by the IP to confirm correct operation. The verification team then verifies the extended range of configuration options."

In addition to IP quality, Stremcha points out another value – assertions mitigate issues in customer design support. He said "Using a fully-described definition of the I/O specification enables us to quickly identify customer usage issues, such as illegal traffic scenarios. With formal assertions, the customer receives immediate feedback from the constraints if they violate the I/O specification."

Grappling with Multiprocessing

ARM’s Dickman pointed to the increasing complexity in key growth areas such as multiprocessing as an example driver for the company’s formal adoption. “Coherent systems based upon MESI-like [Modified, Exclusive, Shared, Invalid] protocols may be a good target for formal verification,” he said.

ARM has used formal verification as a late-stage back-up in some projects for a number of years, but has ramped its efforts in formal over the last six months in the hope of reaping a greater ROI. “Using assertions to capture design intent in the design phase and then using them later in simulation is an accepted best practice in ARM. For a number of years, some project teams have applied varying levels of effort later in the flow to exhaustively verify those assertions formally, with varying results. However, we’re now exploring more in higher-level assertions because, for example, liveness and safety properties have a potentially higher ROI. For instance, ensuring that a circuit is deadlock-free has a very high value.”

What About Productivity?

The feedback on productivity varies a great deal. A perception that users must be truly expert in order to derive productivity advantages from formal is not robustly supported by this feedback. The results look like a situation-dependent mixed bag.

HP’s Lacey says that his team has realized a productivity value – an increase in simulation speed by reducing the number of assertions used in dynamic verification. “We encourage our designers to write assertions in order to communicate design intent. However, that slows simulation. To mitigate the problem, we identify which assertions we can prove in an under-constrained environment; then we omit them from simulation because we know they can never fire. Our strategy right now is to first understand where we can derive the most value from formal tools, and then use them to make our simulation cycles more productive.”

SLE’s Stremcha says that “formal verification requires a great deal of work that should be done in design, anyway. As designers increasingly employ assertions during the design stage, the effort to apply formal decreases.”

ST’s Haller sees occasional productivity boosts because designers must no longer devise block-level testbenches. Nonetheless, the teams must continue to use dynamic verification because formal verification tools still suffer from capacity shortfalls.

According to Infineon’s Galpin, his group uses any productivity gains achieved by formal to apply more effort elsewhere in the verification flow. He said “Formal rarely reduces downstream dynamic verification effort because we tend to apply the same number of dynamic tests to IP as we would have done had we not used formal; and we dynamically verify the assumptions that we made in formal about how the IP will be used. In any case, the confidence established by formal enables the team to devote even more cycles to the core behaviors in the dynamic part of the testbench – particularly in the case of blocks that are likely to be extensively reused.”

According to Ho at DESRES, “Formal verification effort is often equal to or greater than simulation effort. So, we have to use it to make simulation simpler, shorter or unnecessary. That’s where we win the productivity.”

Alcatel-Lucent’s Knäblein estimates that productivity is about the same for formal and dynamic verification. Nonetheless, block-level productivity gains can be quite substantial. In today’s edition of SCDsource [6], Knäblein and his colleague, Hans Sahm, report how they used both automated assertion generation and automated formal methods to verify a complex HW/SW interface in a large SDH/SONET chip – and slashed verification time and effort by 70 percent versus simulation.

What About Project Risk?

Simulation has long been the more-or-less predictable verification route. The technology limitations of formal verification and the consequent uncertainty of success have historically fuelled a reluctance to adopt it.

However, according to IBM's Baumgartner, "The ability to share assertions across formal and dynamic ABV has made an impact on formal adoption. Assertion-sharing allows us to set up a formal testbench, and to redirect the assertions to a dynamic environment if the formal tool can't cope. The effort to establish these formal assertions is thus not wasted. Eliminating the project risk associated with potentially wasted effort in formal enables us to push it to the limit." He continued, "As formal technology continues to improve, we can tackle verification challenges that used to be impossible in formal. For example, we can set up a reusable formal methodology to verify error-code correction – we don't need CPU months of dynamic effort, which itself still carries the risk of verification gaps."

And Methodology?

The importance of methodology came up in almost every interview. So, how do 16 of our 19 respondents rate the methodology support that they receive from technology providers? The responses are shown in figure 6.

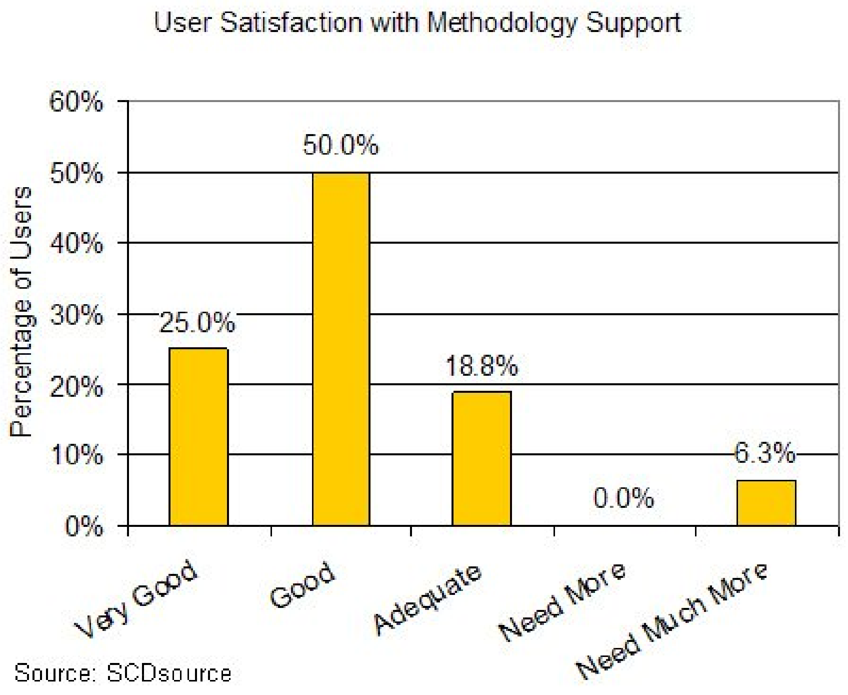


Figure 6: User satisfaction with technology providers' methodology support (Source: SCDsource)

Formal Verification - Where?

What are the sweet 'n' sour spots of formal verification? Clearly, the answers to these questions go to the heart of mixing formal and dynamic methods. Our user respondents broadly agreed on the sweet spots – control circuits, and datapaths with high concurrency that do not involve data transformations. The sour spot is datapath circuits that *do* involve data transformations.

A DVCon 2006 paper [7] spells it out. Example sweet spots include:

- Arbiters of many different kinds
- On-chip bus bridge
- Power management unit
- DMA controller
- Host bus interface unit
- Scheduler, implementing multiple virtual channels for QoS
- Clock disable unit (for mobile applications)
- Interrupt controller
- Memory controller
- Token generator
- Credit manager block
- Standard interface (for example, PCI Express)
- Proprietary interfaces

According to the paper, example sour spots include:

- Floating point unit
- Graphics shading unit
- Convolution unit in a DSP chip
- MPEG decoder

However, as IBM's Baumgartner previously noted, the company *can* formally verify FPUs. In a DATE 2005 paper [8], Baumgartner *et al* report the fully-automated, exhaustive formal verification of fused-multiply-add (FMA) FPUs without the need for customized tools. The approach focuses on the arithmetic correctness of a single arbitrary instruction, and compares the FPU implementation with a simple reference model derived from the processor's architectural specification. The approach uses a combination of case-splitting and automatic model reduction techniques, and isolates the multiplier for dedicated verification. According to the paper, the approach is portable to simulation, emulation, semi-formal, and formal verification.

The foregoing sweet spot list does not include the verification of CPU and application-specific processors or, at least, the parts of those processors that do not perform data transformations. However, formal is being used increasingly to verify such processors.

For example, Infineon used formal to verify its 40-instruction PPv2 network processor [9]. It required only 40 high-level (operation) properties to verify the correct pipelined processing of multiple instructions; the correct operation of permissible, but unpredictable, behaviors such as traps and interrupts; data paths with complex bit-manipulations; and independent execution of multiple threads under all possible combinations of instructions, thread switches, traps and interrupts. An example of a bug detected by formal: an error caused instruction words to be modified while stored in the context switch buffers. The verification team hadn't anticipated this situation. Dynamic ABV might have detected it, but only with a great deal of luck.

DESRES designs ASICs that incorporate custom processors. Ho said "Simply by breaking the verification task down to smaller units, we can very quickly formally verify that an individual processor operation behaves exactly as specified for all possible combinations of inputs. We still have to simulate the whole processor, but the simulation of the formally-verified pieces is essentially "off the table" as far as the verification plan is concerned."

Eliminating the Mystique

Some respondents think that formal verification is shrouded in an unwarranted mystique (see our **cartoon** at the end of part 1).

Ho at DESRES said "Formal is just another tool in the toolkit. We have the objective to mainstream formal in 2009. Our goal is to have every member of the verification team be proficient in the use of both simulation and formal in that timeframe."

Infineon's Galpin said "There's a common perception that you have to be a mathematical genius to use formal. It's not nearly as difficult to use as it's portrayed." He also had a suggestion for conference organizers: "I think that conferences could improve this perception by focusing more on how formal is used in practice, and clearly differentiate this practice from the advanced algorithm development presented by the academics."

We at SCDsource hope that the results of our survey will help to eliminate at least some of that mystique.

In part 2 of the STR, we discuss:

- The detailed results for the 17 use cases.
- How formal is currently being used with dynamic verification.
- The application of formal in the ESL space.
- How technology users and providers envisage formal methods in 2012.

References

1. The Art of Verification with SystemVerilog Assertions. Faisal Haque, Khizar Khan, Jonathan Michelson. Verification Central 2006.
2. Practical Approaches to Deployment of SystemVerilog Assertions. Faisal Haque, Jon Michelson. EE Times 2007.
3. Structured Assertion Design Verification for Complex Safety-Critical Hardware. Kristoffer Karlsson, Håkan Forsberg. Military and Aerospace Programmable Logic Devices (MAPLD) Conference 2008. Paper and Presentation.
4. A comparison of three verification techniques: directed testing, pseudo-random testing and property checking. Mike Bartley, Darren Galpin and Tim Blackmore. Proceeding of the 39th Design Automation Conference, 2002.
5. Early formal verification of conditional coverage points to identify intrinsically hard-to-verify logic. Richard Ho, Michael Theobald, Martin M. Deneroff, Ron O. Dror, Joseph Gagliardo, David E. Shaw. Proceedings of the 45th Design Automation Conference, 2008.
6. Automated formal method verifies highly-configurable HW/SW interface. Joachim Knäblein and Hans Sahm. SCDsource 2009.
7. Guidelines for creating a formal verification testplan. Harry Foster, Lawrence Loh, Bahman Rabii, Vigyan Singhal. DVCon 2006. Paper and Presentation.

8. Automatic Formal Verification of Fused-Multiply-Add FPUs. Christian Jacobi, Kai Weber, Viresh Paruthi, Jason Baumgartner. Proceedings of DATE 2005.

9. Achieving Certified IP Quality Efficiently. Lorenzo di Gregorio, Carlo del Giglio, Michael Siegel. EE Times 2007.

10. Zero Escape Plans: Tying Together Design, Simulation, and Formal Methods for Bulletproof Stepping Validation. Erik Seligman, Carl Dreyer, Ken Haren, Raman Nayyar. Proceedings of DVCon 2008. Reported in "Formal Verification Expands Its Use Model" by Bill Murray, SCDsource 2008.

11. Post-Silicon Debug Using Formal Verification Waypoints by C. Richard Ho, Michael Theobald, Brannon Batson, J.P. Grossman, Stanley C. Wang, Joseph Gagliardo, Martin M. Deneroff, Ron O. Dror, David E. Shaw. Proceedings of the 43rd Design Automation Conference, 2006.

12. Can We Really Do Without the Support of Formal Methods in the Verification of Large Designs? Umberto Rossi. 42nd Design Automation Conference, 2005.

13. Assertion-Based Design. Harry Foster, Adam Krolnik, David Lacey. Springer Verlag, 2004.

14. Unified Coverage Database Application Programming Interface (API) Design Specification Document, Draft 29, April 2, 2009. Accellera Unified Coverage Interoperability Standard (UCIS) committee.

15. Formal Techniques Speed Up Interconnect Verification of SystemC Virtual Platform Models. Wolfgang Ecker, Volkan Esen, Robert Schwencker, Thomas Steininger, Michael Velten. DVCon 2008. Reported in "Formal Verification Expands Its Use Model" by Bill Murray, SCDsource 2008.

Further Reading

1. Formal Property Checking - What The Users Say, by Richard Goering, SCDsource 2008.

2. Increasing Confidence of Complex Hardware in Safety-Critical Avionics Using Formal Methods. Kristoffer Karlsson and Håkan Forsberg. Military and Aerospace Programmable Logic Devices (MAPLD) Conference 2006.

Special Technology Report

Mixing Formal and Dynamic Verification, Part 2

By Bill Murray

In part 1 of this STR, we:

- Quantified the broadening and deepening of formal verification deployment
- Quantified the fast ramp of recent adopters
- Quantified the advancement of formal verification into design
- Discussed the crumbling barriers to formal's adoption
- Chronicled the quest for quality
- Discussed productivity
- Outlined formal verification's sweet 'n' sour spots

And we published our Exhaustive Proof cartoon.

In part 2 of this STR, we:

- Outline the 17 use cases used in the survey
- Discuss the detailed survey results
- See how formal is currently being used with dynamic verification
- Look at the application of formal in the ESL space
- Hear from both technology users and technology providers how formal methods might look in 2012
- Wrap up with the most significant findings in both parts of this report

The SCDsource Survey

According to Gary Smith of Gary Smith EDA, formal property checking is used by 65 percent of chip design companies. Smith said "The proliferation of SystemVerilog and its assertion language, SVA, has boosted the adoption of formal property checking."

We surveyed 19 engineering managers and engineers in 16 companies to quantify and characterize their use of formal property checking. Our survey population spans the design and verification of application-specific chips, general purpose processors, graphics processors and silicon intellectual property (IP) for applications in computing, consumer electronics, medical electronics, military/aerospace, networking and wireless communications.

The 16 companies are industry-leading IP, chip and systems design companies – Alcatel-Lucent, Analog Devices, ARM, Cisco, DE Shaw Research (DESRES), Fujitsu Microelectronics Europe, HP, IBM, Infineon, Intel, Nvidia, Qualcomm, Saab, Silicon Logic Engineering (Tundra), STMicroelectronics and Sun Microsystems.

Of the 19 respondents, 14 were employing formal verification in 2006 – this STR refers to them as established users. This STR refers to the other five respondents, who adopted formal after 2006, as recent adopters.

The 17 Use Cases

The survey covered the following use cases. These use cases were gleaned largely from discussions with our interviewees, as well as from published material.

1. Early RTL code analysis to detect common design errors. Uses formal code checks and/or assertions generated either manually or automatically [5].
2. Early block-level sanity checking and bring-up verification to check micro-architecture and implementation intent, using white-box assertions.
3. Share block design intent and integration assumptions, using assertions.
4. Capture and share coverage targets using assertions that validate formal constraints and define functional coverage points.
5. Protocol compliance verification.
6. Exploration and bug hunting for block/subsystem features, functions, operations, and transactions, using grey-box or black-box assertions.
7. Exhaustive verification of block/subsystem features, functions, operations, and transactions using grey-box or black-box assertions.
8. Verification plan-driven systematic, thorough block/subsystem verification.
9. Analysis of assertion sets for missing assertions – detects verification gaps and validates/improves informal specification.
10. Analysis of coverage items not reached by simulation/testbench.
11. Subsystem bring-up and integration verification.
12. Signal connectivity verification.
13. Engineering change order (ECO) verification.
14. Root cause analysis of complex errors found by simulation.
15. Root cause analysis of complex errors for post-silicon debug [10, 11].
16. Legacy block/IP exploration and redesign using assertions.
17. Functional sign-off criterion.

Survey Results

We show some of the more important survey results below. Note that all survey data in the following graphs are rounded, while derived data (such as growth rates and ratios) are calculated from the original, non-rounded survey data. The resulting tiny inconsistencies in the maths yield considerably greater clarity in the analysis.

Use Case Employment in 2009

Use case employment in 2009 by our 19 respondents is shown in figure 7. The results are ordered from most widely used at the top to least widely used at the bottom.

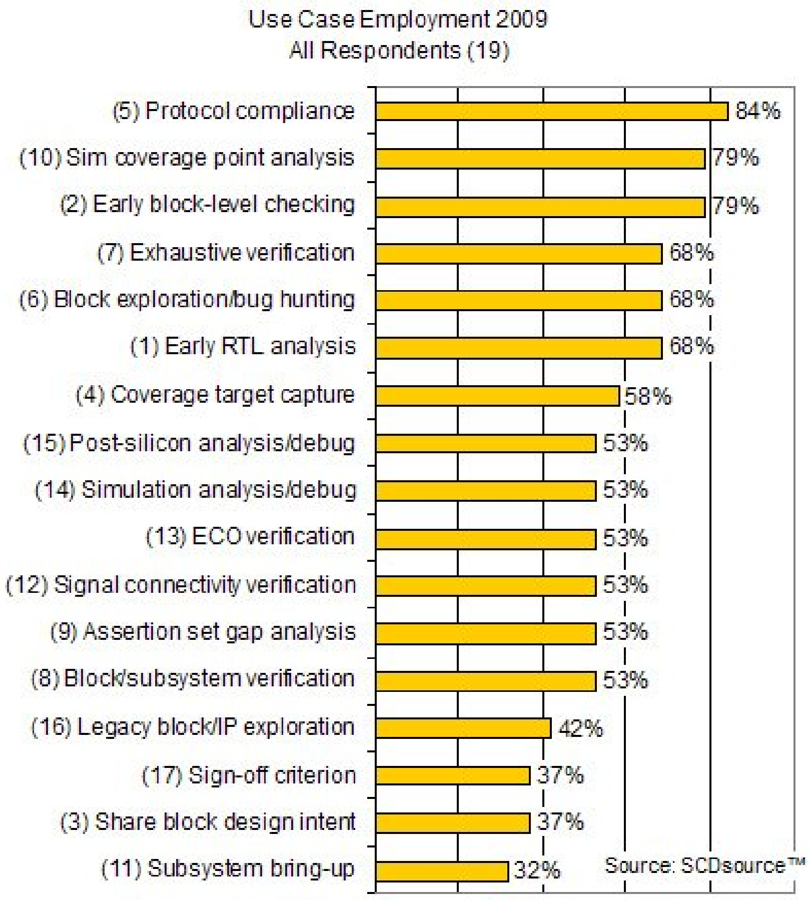


Figure 7: Use case employment 2009 (Source: SCDsource)

The results show significant use of formal verification throughout the design and verification flow from initial RTL checks to verification sign-off, and even after first silicon availability. Even the least-used case is employed by nearly one-third of our respondents. Of particular note is:

- The evidence that formal verification is advancing backwards into design (see part 1 of the STR). Early RTL analysis is already among the most widely-employed use cases in 2009, used by 68 percent of the respondents; and about 37 percent use formal assertions to define and share design intent.
- Analysis of coverage items not reached by simulation is a second-ranked use case, with 79 percent of respondents employing it. Clearly, this complementary verification approach is a key value in mixing formal and dynamic verification.
- More than half of the respondents use formal analysis to identify the root cause of complex errors in post-silicon debug.

Top 3 Use Cases in 2009

Does "most widely used" necessarily mean "most important?" We asked our survey participants to identify their Top 3 use cases. The results are shown in figure 8, maintaining the use case order of figure 7.

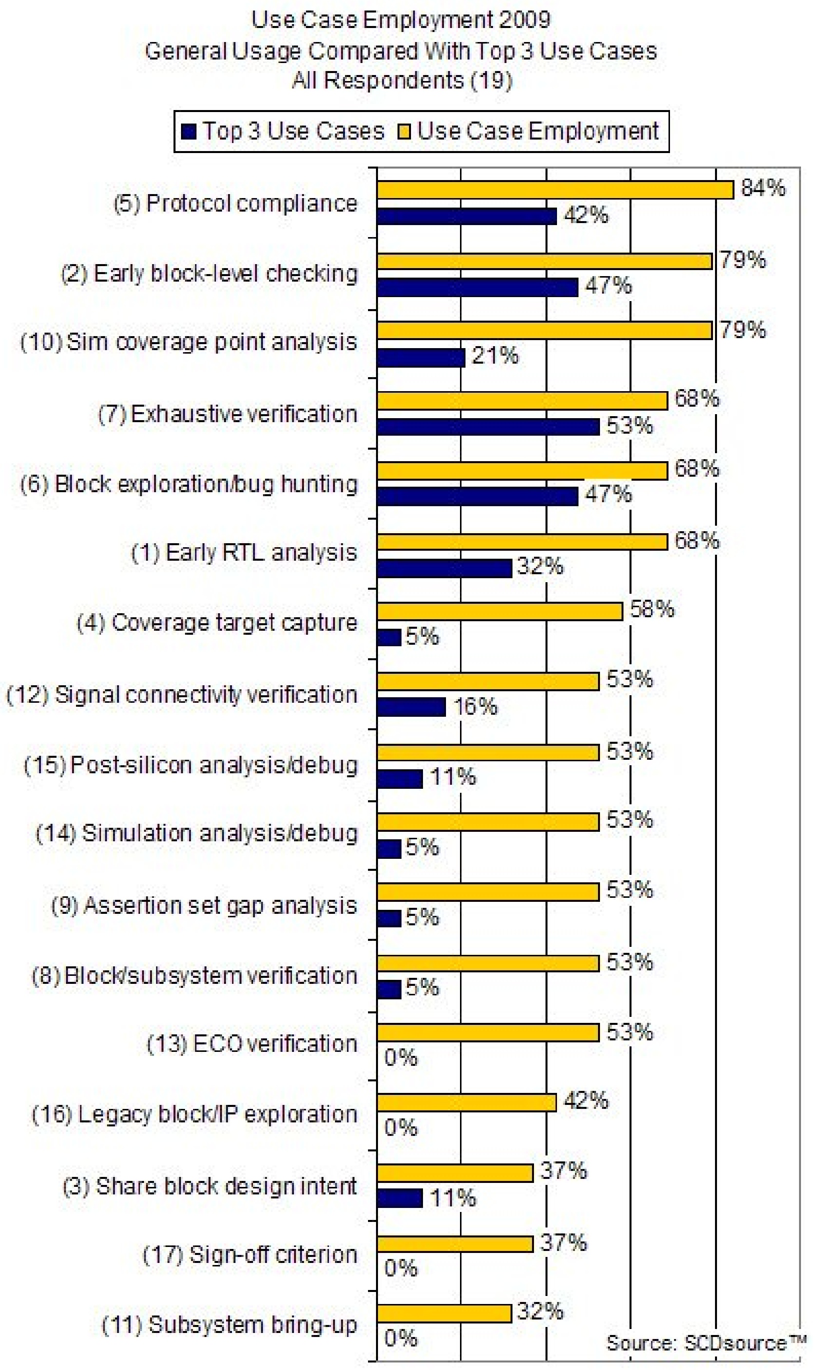


Figure 8: Aggregate Top 3 use cases vs. use case employment 2009 (Source: SCDsource)

The results illustrate the importance of the question. Protocol compliance verification is the most widely employed use case, employed by 84 percent of respondents, but only half of those respondents identified it as a Top 3 use case. By contrast, exhaustive verification is the fourth most widely employed use case, used by 68 percent of the respondents, but nearly four-fifths of them – 53 percent of respondents – identified it as a Top 3 use case. Result: protocol compliance verification ranks fourth in the Top 3 ranking, while exhaustive verification ranks first. Exhaustive verification – formal's major traditional perceived strength – remains the most important use case, despite formal's proliferation up and down the design and verification flow.

Two use cases tie for second rank, with 47 percent of respondents placing these use cases in their Top 3:

- Early block-level sanity checking and bring-up verification to check micro-architecture and implementation intent, using white-box assertions.
- Exploration and bug hunting for block/subsystem features, functions, operations, and transactions, using grey-box or black-box assertions.

Note: protocol compliance verification is not ranked third because the second rank tie puts three use cases ahead of it – there is no third rank.

Is there a difference in use case employment between recent adopters and established users in 2009? The two are compared in figure 9, maintaining the use case order of figure 7.

The shapes of the two distributions are similar, indicating that the two groups may have similar assessments of the ROI of any given use case.

Interestingly, of the recent adopters, 100 percent employ formal methods to analyze coverage items not reached by simulation, compared with 71 percent of established users. The absolute percentages, however, should be used with caution – a single “vote” in the recent adopter population can change the use factor by 20 percentage points, while a single “vote” in the established user population can change it by only about 7 percentage points.

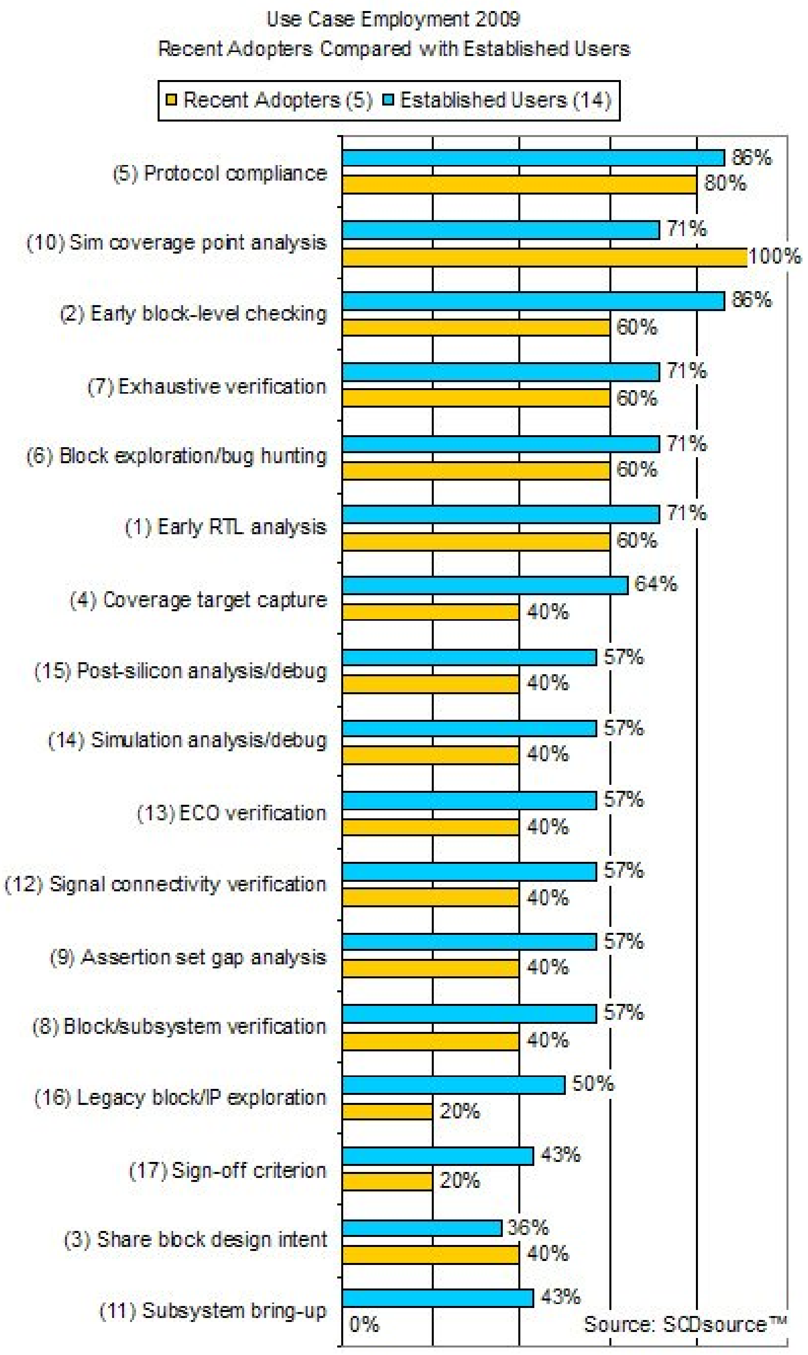


Figure 9: Use case employment 2009 recent adopters vs. established users (Source: SCDsource)

Adoption Speed and Profile

In part 1 of this STR, we compared the average use case employment of recent adopters and established users, and observed that the latter appear to be leading the former by about three years. Specifically, in 2009, recent adopters are employing about 7.8 use cases on average versus 6.6 by the established users in 2006; while recent adopters are projected to use 10.8 use cases in 2012 versus 10.4 by established users in 2009.

The obvious question is: how do the detailed employment profiles compare at similar stages of adoption? Are there any obvious similarities or dissimilarities? A comparison of the two groups' use case employment is shown in figure 10, maintaining the use case order of figure 7. It compares use case employment by established users in 2006 with that of recent adopters in 2009.

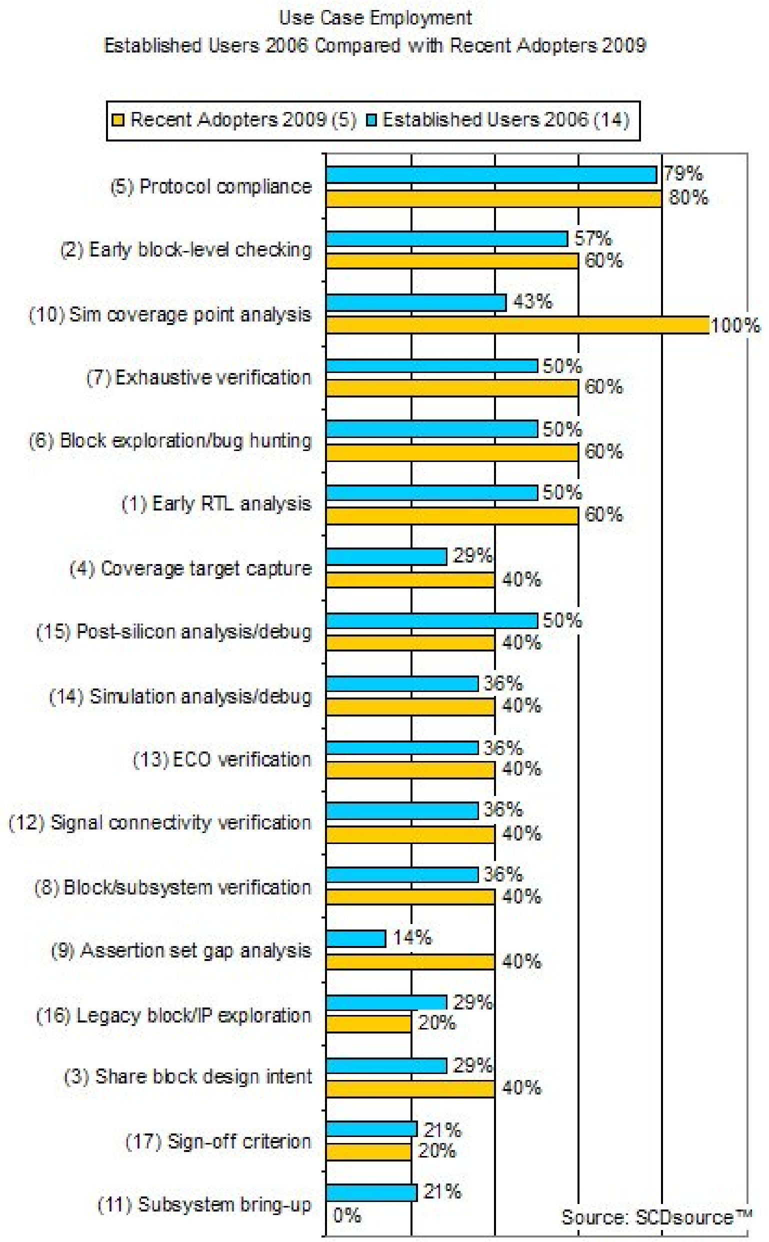


Figure 10: Established users 2006 compared with recent adopters 2009 (Source: SCDsource)

With the exception of the analysis of coverage items not reached by simulation, the shapes of the use profiles are similar. Moreover, the percentage of respondents in each group employing any given use case is similar – often nearly identical – with the exception of subsystem bring-up and the analysis of coverage items not reached by simulation.

In summary, then, the recent adopters in 2009 are in roughly the same state of adoption as the established users were in 2006. Could improved methodologies and technologies narrow the gap or, more accurately put, could they significantly accelerate the adoption of formal verification? If so, how? That is the subject of the next section.

Mixed Formal and Dynamic – How?

In a DAC 2005 paper [12], ST's Umberto Rossi observed that "From the user point of view, verification products still lack integration between dynamic analysis and formal assertion analysis. A closer cooperation or a better merger of the respective results may constitute a significant plus for the users."

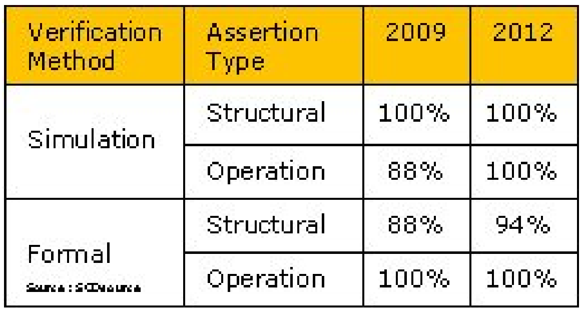
This is the key to greatly increased formal verification use in the mainstream verification flow. Knowing where best to apply formal and dynamic verification methods and what the coverage/productivity payback is in any given scenario will help design and verification teams to make dependable, highest-ROI decisions with predictable results. Such consistent, comprehensive, objective data could identify and help proliferate best practices – and narrow the gap between recent adopters and established users.

Cross-Leveraging Assertions

So, how does the industry achieve that integration between formal and dynamic verification? Of course, both techniques leverage assertions. What kind of assertions, and where? In our survey, we partitioned assertions into two basic classes [13]:

1. Structural assertions
    - Low level assertions that verify detailed RTL behavior, such as that of a finite state machine (FSM).
    - Adequate for localized sanity checking and verification.
    - Must often be modified in response to RTL implementation modifications.
    - Difficult-to-impossible to verify a design exhaustively.
2. Operation/transaction assertions
    - Assertions that verify RTL behavior at a higher level, such as block reads and writes.
    - Correspond more closely to functionality as defined in the specification – a more intuitive approach to verification.
    - Must often be modified in response to specification changes, but generally not to RTL implementation changes.
    - Can deliver significantly higher verification productivity than structural assertions.
    - Suitable for exhaustive verification.

Our respondents' use of these assertions in simulation/testbenches and formal verification is shown in figure 11.



Verification Method	Assertion Type	2009	2012
Simulation	Structural	100%	100%
	Operation	88%	100%
Formal	Structural	88%	94%
Source: SCDsource	Operation	100%	100%

Figure 11: The use of different assertion types in simulation and formal verification (Source: SCDsource)

All of our respondents to this part of the survey (in this case, 16, not 19) use structural assertions in simulation, and all use operation assertions in formal. Eighty-eight percent of respondents use operation assertions in simulation, projected to increase to 100 percent by 2012; and 88 percent use structural assertions in formal, projected to increase to 94 percent by 2012.

This large degree of overlap would indicate the potential for a great deal of cross-leverage. So, how are they being cross-leveraged? Some respondents employ the same assertions in both environments, but many don't. Moreover, our survey shows that only 31 percent of respondents use formal to verify testbench assertions, both structural and operational. However, this is projected to increase to 63 percent by 2012.

Integrating Coverage Metrics

Establishing a tighter coupling between simulation and formal verification is clearly a significant need for most of the respondents in the survey, but they need a methodology to do it. Right now, most of our respondents mix formal and dynamic verification at only the verification plan level. Thereafter, the two approaches largely go their separate ways. Each has its own coverage goals; progress is measured or estimated separately; and total coverage is determined by eye-balling overall progress to the verification plan.

ARM wants to make eye-balling easier. Bryan Dickman said "We're moving towards database-driven planning. The idea is to upload results to a mySQL database and use the verification plan as a database query. We would then have a live view of results versus plan. Of course, both plan and database would contain both simulation and formal targets. That's our destination."

However, what everyone really wants is a unified coverage approach. DESRES' Ho said "A combined formal and simulation verification method would be much more effective if users could know how much coverage was achieved by each technique, and thereby avoid redundant verification." He continued "We need the ability to import coverage data from different tools from different vendors. Currently, there is no good way of doing that without a great deal of user involvement in translation."

Ho is now the chair of Accellera's Unified Coverage Interoperability Standard (UCIS) committee. The stated objectives of the committee are to:

- Identify interoperability opportunities between various coverage sources
- Define standard coverage models for commonly used metrics
- Define an operability standard that allows coverage data to be exchanged among EDA vendors' tools and IC vendors environments

Ho, together with Accellera's chair, Shrenik Mehta, reported progress-to-date. Ho said "The committee seeks to establish standards for those coverage items upon which members can agree, but also to define an extension mechanism for vendor-specific or user-specific coverage types." Ho observed that "Standardizing coverage types will enable the automatic merging of data, while standardizing an extension mechanism for new coverage types allows for innovation."

A draft proposal [14] dated April 2009 identifies three use models for data merging, 16 verification coverage types targeted for standardization, and a conceptual API for the database.

The use models cover:

- Temporal merging: merging of coverage data across multiple runs of the same verification process.
- Spatial merging: merging of coverage data across different parts of a design.
- Heterogeneous data merging: merging of data from different verification processes.

The 16 verification coverage types are:

- Toggle coverage
- Line/statement coverage
- Branch coverage
- Condition/expression coverage
- Trigger coverage
- FSM state coverage
- FSM transition coverage
- Value-transition coverage
- Path coverage
- Cover point/coverage point
- Cross-coverage
- Cover group coverage
- Cover property
- Assertion coverage
- Assertion result
- Transaction coverage

Accellera expects to release a document to the general public in 2010.

System-Level Formal Verification

Formal Analysis

In his DAC 2005 paper [12], ST's Umberto Rossi noted that, in order to verify the millions of complex configurations of a parametric IP "Formal methods have to extend their effectiveness by including so-called transactional level models (TLM) in their analysis, which abstract the mechanisms that SoC blocks use to communicate and synchronize among themselves. Two major problems have to be faced: the first is that TLM includes a range of abstraction levels and the second is that a mapping between TLM assertions and RTL assertions should be provided and formally proven."

Three years later, not much had changed. In a DVCon 2008 presentation [15], Wolfgang Ecker, principal engineer in Infineon's communications solution business group, observed "The barrier to SystemC TLM interconnect verification is that SystemC does not support native temporal assertions. Previous efforts to rectify this deficiency in SystemC apply assertion-based verification (ABV) only at the RT-level."

Mike Meredith, president of the Open SystemC Initiative (OSCI), said "There have been discussions in the OSCI verification working group about what temporal assertion checking at the TLM level really means. No firm consensus has been reached as to any standardization. However, there are OSCI members who are investigating formal property checking in SystemC, and they are making some progress in devising methods to do it. These would be proprietary, but are still valuable additions to the SystemC ecosystem."

Does OSCI expect to see any progress by 2012? Meredith said "As far as development and deployment in the ecosystem is concerned, I certainly do. As far as standardization is concerned, it depends upon what makes sense to the developers of the proprietary approaches and the rest of the verification working group about what to try to standardize."

System-level Equivalence Checking

However, there is one system level formal verification approach that *is* in use today. System level equivalence checking verifies equivalence between a C/C++/SystemC architecture model and the corresponding RTL micro-architecture implementation. The technology's sweet spot is assertion-based formal verification's sour spot – data transformation logic.

According to Prosenjit Chatterjee, hardware engineering manager at Nvidia, "Graphics processors leverage a broad range of data transform functions, such as arithmetic logic, address translation and, in general, complex mux logic. Many of these blocks are quite complex, mission-critical functions – not just simple blocks."

Prior to adopting the technology, said Chatterjee "We relied upon directed and directed random simulation tests to compare the two models. It found a lot of bugs, but it could not exhaustively cover all possible input scenarios, so it was not possible to conclusively prove equivalence. Now, we can prove equivalence with system level equivalence checking, and use any equivalence mismatches to devise simulation tests."

Adoption took a matter of a few days, according to Chatterjee. "There is not much training required. It is not necessary to write properties because, in essence, the C model is the property. The environment requires simple input constraints, such as min/max input values. Sometimes the equivalence check is push-button; sometimes we must decompose the input values into sub-cases or break up the two models at transaction matching internal interfaces, and then prove each one individually."

The value of the approach grows with increasing expertise in writing C models that the technology can handle. According to Chatterjee, "A C model developer might describe data transform functions using dynamic memory allocation or loops with arbitrary bounds. However, hardware design needs fixed bounds, so C developers must replace these constructs with fixed-bound constructs – say, a fixed memory with fixed min/max bit count." In other words, the developer has to use synthesizable code, regardless of whether the design team uses high level synthesis. Chatterjee added "As the technology is increasingly enhanced with further C constructs, developers have more freedom to use constructs that come more naturally to them."

Nonetheless, the requirement that the code be synthesizable remains. Any disadvantages? "C is not really optimal for synthesis. The C model might actually be larger than the RTL model," said Chatterjee. He also pointed to the inherent difficulty of identifying correspondence between the abstract C model and the RTL implementation model. "It's not the same as RTL/gate and gate/gate equivalence checking," he said.

Chatterjee concluded “We plan to use system level equivalence checking on as many data transformation functions as possible. Looking ahead to 2012, I hope to see system level equivalence checking handle much bigger design sizes.”

Asked about the progress towards an agreement on the synthesizable subset of SystemC, OSCI’s Meredith said “We’ve made a lot of progress over the last twelve months. There is a significantly upgraded draft in the OSCI internal approval loop, and we expect to put it into public review this Spring.

The World in 2012

Use Case Employment in 2012

In part 1 of this STR, we showed that use case employment will grow by 32 percent from an average of 9.7 use cases per respondent in 2009 to 12.8 in 2012 – a significant deepening of formal use in this population of 19 respondents. What does the detailed 2012 projected use case employment look like? Results are shown in figure 12, maintaining the use case order of figure 7.

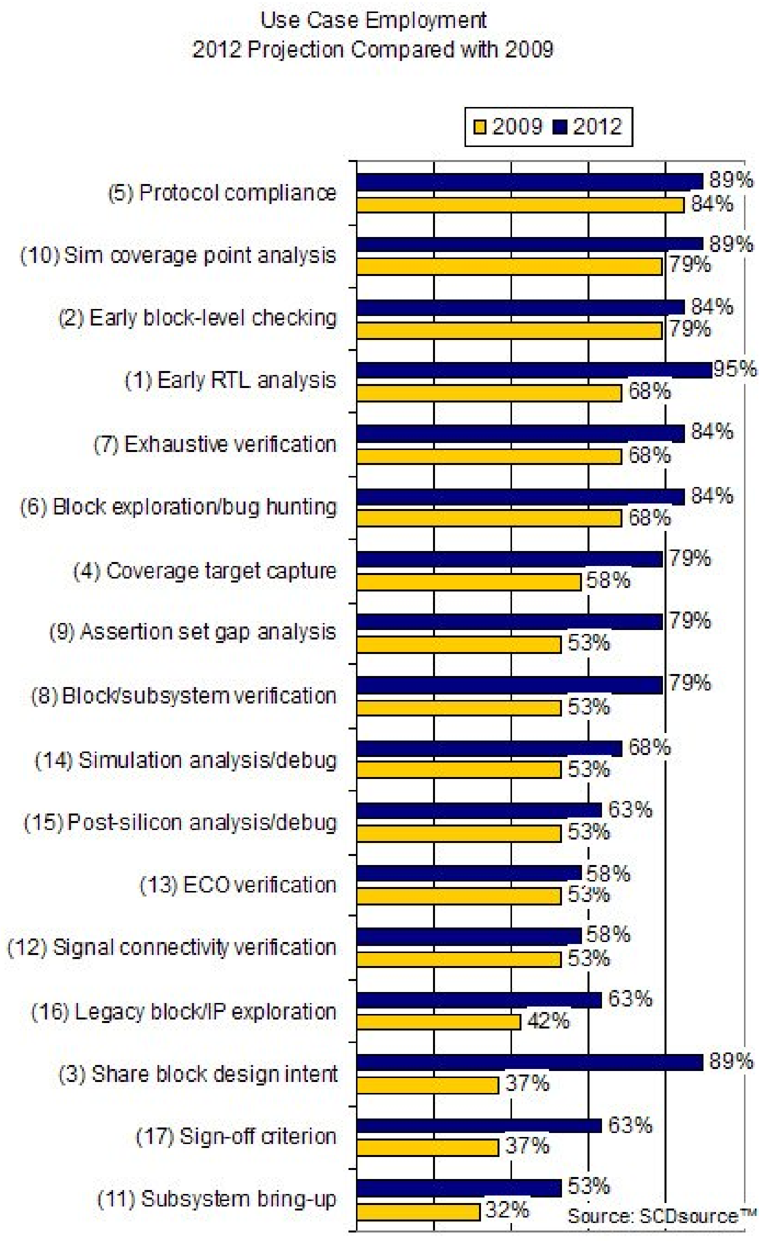


Figure 12: 2012 use case employment compared with 2009 (Source: SCDsource)

The most widely deployed use cases in 2012 are projected to be:

1. Early RTL analysis, identified by 95 percent of respondents. This is up from 68 percent of respondents in 2009. This proliferation and growth (38 percent over 2009) indicates widespread adoption of formal methods by design engineers.
2. Share block design intent and integration assumptions, identified by 89 percent of respondents. Again, this proliferation indicates widespread adoption of formal methods in design.

The top two growth cases from 2009 to 2012 are:

1. Share block design intent and integration assumptions, which is projected to grow by 143 percent from 37 percent of respondents in 2009 to 89 percent in 2012.
2. Sign-off criterion, which is projected to grow by 71 percent. Proliferation of this use case from 37 percent of respondents to 63 percent of respondents is consistent with (a) a growing confidence in formal verification and (b) a consequent reliance upon formal verification at a very critical point in the mainstream verification flow.

Users look to 2012

We asked interviewees to look out three years and visualize what formal verification may look like – or what they would like it to look like. We were not looking for a forecast, just a vision of how the methodology and the technology might look in 2012. Most foresee "bigger, better, faster" technology – more capacity, improved ease of use, faster proofs and debug. And most assume that these improvements will occur incrementally.

However, closer integration of formal and simulation methodologies and technologies was at or near the top of everyone's list. Users pointed to the need for unified coverage metrics, a modular verification infrastructure, common constraints, and a correct-by-construction design flow. And most are keeping a close eye on the Accellera UCIS efforts.

Modular Infrastructure

For HP's David Lacey, the importance of the UCIS effort is more than simply melding coverage results – it is critical to the establishment of a modular verification infrastructure. He said "Such an infrastructure would give us more of a toolbox approach to verification. For example, we would be able to integrate verification IP into formal and simulation in much the same way. So, formal needs to go down that path."

Common Constraints

D. E. Shaw Research's Richard Ho would really like constrained random testbenches to have the same constraints as formal. "This would enable simulation to be really smart. You could run simulation up to a coverage point of interest, then invoke formal analysis. This requires that the constraints for random testbenches be understandable by formal tools. The problem is that testbench writers use biasing of random variables and layered scenarios instead of using assertions to constrain the input sequences. A common language would solve a big interoperability problem and yield very smart verification."

Infineon's Darren Galpin concurs. "When writing constraints for random dynamic verification and formal verification environments, you're essentially doing the same thing, but in a different way. For the random dynamic environment, you're writing a set of code to generate – for example – possible stimuli. You then take a random test path through the protocol space. In the formal environment, you have to constrain the proof to legal operations and test everything simultaneously. I don't see any reason why those two sets of constraints ultimately couldn't be used for both tools."

Correct-by-Construction Design Flow

ARM's Bryan Dickman said "We hope to see a correct-by-construction design flow that may also reduce the overall verification burden. We could get there by having designers use the tools from design onwards in the flow to systematically explore and formally prove their code. At the moment, we're using formal rather retrospectively."

Formal “a must”

The last word from the users goes to IBM’s Jason Baumgartner: "It used to be that you must have sinned greatly to have the formal verification guy in your office. But, with greatly enhanced capacity and reusability, formal has become substantially more mainstream, and is now much easier for casual users to set up. Our post-design ‘lessons learned’ reviews generally find that we should do more in formal and mixed formal/dynamic verification. Many now view formal as a must.”

Technology providers look to 2012

Cadence Design Systems

Sarah Lynne Cooper Lundell, senior product marketing manager, said “By 2012, the focus will be on analyzing and optimizing clear and measurable unified verification metrics. How the contributions to the metrics are generated will be relatively less important. They could come from any combination of formal, hybrid, testbench simulation, acceleration, emulation or other engines. But the discussion will be about what the metrics mean for reaching verification closure.”

According to Lundell, this shift will increase formal’s ROI because “unified metrics eliminate the need to duplicate efforts to verify any given functionality, and allow project teams to deploy the right solution for a given task and for the skills available. Traditionally, formal analysis has been used either early in the verification process prior to the availability of a testbench, or on really tough verification targets; but generally adjacent to mainstream verification. Unified metrics will allow project teams to track formal’s progress alongside that of testbench simulation. This will break down walls that have hampered formal’s adoption.”

Calypto Design Systems

Anmol Mathur, CTO, expects system-level equivalence checking to become a standard step in high-level synthesis (HLS) verification flows by 2012. He said “The two main enablers will have been established. First, the synthesizable subsets of C/SystemC will have been standardized; and second, a standard exchange format to transfer information from HLS tools to system level equivalence checking technology will have been agreed.”

Mathur continued “This will enable checking tool to leverage information about boundaries such as function hierarchies or loop bodies to decompose the complex formal equivalence checking problem in order to scale to the block sizes that HLS tools will target in 2012.”

Moreover, he said, “Fundamental progress in formal solvers that work at the operator-level rather than manipulating bit-level Boolean functions has been critical to allowing system level equivalence checking technology to tackle arithmetic intensive designs. I expect better merging of this technology with formal analysis of control to further boost tool capacity. And further improvements in sequential proof engines will allow designs with larger divergence in sequential behaviors to be checked.”

Jasper Design Automation

Jasper sees formal methods extending beyond verification. Rajeev Ranjan, CTO, said “Higher tool capacity and scalability, coupled with innovative methodology, will enable the application of formal throughout the design flow, from architectural analysis, through design and integration, to silicon debug.”

Ranjan continued “And formal technology will bring breakthrough changes in IP delivery. Third party IP will be encapsulated in a formal technology-based ‘executable shell’ that will make it easy for designers to adopt the IP. It will allow designers to easily comprehend the critical behaviors of the IP, explore dependencies, and understand bus protocols.”

The company also projects that formal technology will be applied at the system level. Ranjan said “Standard ways of architectural and system-level modeling will enable the application of formal to property verification of high-level models, such as SystemC models. We shall see formal methods used in architectural models, TLM, HW/SW co-design, and algorithms developed in MatLab.”

Mentor Graphics

Harry Foster, chief verification scientist, foresees new, automatic applications built on formal technology that are easy to use and require no special expertise in formal verification or languages. “Furthermore, we predict a tighter integration of formal and simulation algorithms to solve specific problems in coverage closure as well as pre- and post-silicon debugging,” he said.

Foster pointed out that about 19 percent of the industry has adopted formal property checking, and about 41 percent has adopted simulation-based, coverage-driven techniques that leverage functional coverage properties specified using modern assertion languages.

He said “Clearly, there are opportunities to improve verification processes by taking advantage of assertions in either simulation or formal property checking. However, part of the challenge of adopting assertion languages is the expertise required to grasp temporal logic and declarative forms of specification. These concepts, which form the foundation of today’s assertion languages, are foreign to many of engineers. Perhaps the recent emergence of assertion libraries and assertion-based verification IP will assist the adoption of assertion-based techniques in the future.”

OneSpin Solutions

Michael Siegel, OneSpin's director of product marketing, stated “Today, formal verification is often used in addition to dynamic verification to increase design quality. However, the main potential of combining formal and dynamic lies in mastering the verification productivity challenge. We foresee enhanced methodologies and tools that guide users to the optimal verification approach in order to predictably reduce overall verification effort. Knowing when, where and how to apply the most appropriate method – formal or dynamic verification – is key.”

Siegel foresees enhancements that will spawn more automated and integrated formal/dynamic verification flows. “Formal will be a standard, mainstream technology in the verification toolkit for design, verification and integration engineers,” he said, “and enhanced tool interoperability will reduce the manual effort required in today’s combined formal/dynamic verification flows.”

“Increased verification productivity also depends upon the ability to accurately assess overall verification coverage and tool support to close coverage gaps,” Siegel continued. “So, it is critical that the industry establishes suitable standard coverage metrics shared between formal and dynamic.”

Real Intent

Carol Hallett, Real Intent's vice president of sales and marketing, said that automatic formal verification is an essential component in all functional verification flows today, and that its use will increase substantially by 2012.

According to Hallett, “Automatic formal verification applies pre-determined rules to perform error checking automatically, with virtually no engineering effort or design style restrictions. It runs formal analysis automatically, and generates failure reports that minimize debug effort.” She continued “The methodology tackles two endemic problems. Its early bug detection mitigates the spiral of bug detection, debugging and bug fixing; and its automation deals with the ever-growing problem of constrained resources.”

Hallett predicts that improved automatic error checking rules applied early in the design flow; faster convergence on larger designs; and much more advanced reporting and debugging support will increase the bug detection rate from about 50 percent today to about 70 percent by 2012.

Synopsys

Dan Benua, principal corporate application engineer, sees a continuing trend to use more formal and assertion-based verification to complement the dominant constrained-random coverage-driven simulation approach. “This trend is being driven by the need to improve verification cost-efficiency, and is also facilitated by progress in formal technology, language and library standardization, and easier-to-use tools,” he said

He continued, “There will still be experts doing hard-core model checking, but the average designer or simulation user will be leveraging more formal analysis also. In some cases, engineers may not even be aware that formal algorithms are running inside their simulator; providing benefits like debugging hints, coverage closure, or smarter stimulus generation. Multi-core hardware runs these activities in the background with minimal impact on simulation runtime.

“We are still in the early stages of understanding how to integrate and potentially merge these technologies, said Benua. “We see big opportunities ahead to improve the functional verification process and we are hard at work on the technologies, methodologies, and tools needed to achieve them.”

Wrap-Up

So, what are the major findings of this STR?

- In this population of 19 users, formal verification’s use nearly doubled from 2006 to 2009.
- The 5 recent adopters are closely tracking the adoption curve of the 14 established users.
- The results show significant use of formal verification throughout the design and verification flow from initial RTL checks to verification sign-off, and even after first silicon availability.
- Adoption barriers are crumbling – capacity increases and ease-of-use enhancements make adoption easier – but many respondents want to see a lot more progress.
- The primary reason for adopting formal was verification quality – but productivity is a key factor in the extent of formal’s deployment.
- Formal’s sweet spots are expanding.
- Formal is “a must.”
- System level property checking is in a barely embryonic stage, but system level equivalence checking has been established as a viable approach.
- Last, but not least, we still need a robust, systematic methodology for mixing formal and dynamic verification. Our interviewees view the work of Accellera’s UCIS as critical to this effort.

References

1. The Art of Verification with SystemVerilog Assertions. Faisal Haque, Khizar Khan, Jonathan Michelson. Verification Central 2006.
2. Practical Approaches to Deployment of SystemVerilog Assertions. Faisal Haque, Jon Michelson. EE Times 2007.
3. Structured Assertion Design Verification for Complex Safety-Critical Hardware. Kristoffer Karlsson, Håkan Forsberg. Military and Aerospace Programmable Logic Devices (MAPLD) Conference 2008. Paper and Presentation.
4. A comparison of three verification techniques: directed testing, pseudo-random testing and property checking. Mike Bartley, Darren Galpin and Tim Blackmore. Proceeding of the 39th Design Automation Conference, 2002.
5. Early formal verification of conditional coverage points to identify intrinsically hard-to-verify logic. Richard Ho, Michael Theobald, Martin M. Deneroff, Ron O. Dror, Joseph Gagliardo, David E. Shaw. Proceedings of the 45th Design Automation Conference, 2008.
6. Automated formal method verifies highly-configurable HW/SW interface. Joachim Knäblein and Hans Sahm. SCDsource 2009.
7. Guidelines for creating a formal verification testplan. Harry Foster, Lawrence Loh, Bahman Rabii, Vigyan Singhal. DVCon 2006. Paper and Presentation.
8. Automatic Formal Verification of Fused-Multiply-Add FPUs. Christian Jacobi, Kai Weber, Viresh Paruthi, Jason Baumgartner. Proceedings of DATE 2005.
9. Achieving Certified IP Quality Efficiently. Lorenzo di Gregorio, Carlo del Giglio, Michael Siegel. EE Times 2007.
10. Zero Escape Plans: Tying Together Design, Simulation, and Formal Methods for Bulletproof Stepping Validation. Erik Seligman, Carl Dreyer, Ken Haren, Raman Nayyar. Proceedings of DVCon 2008. Reported in Formal verification expands its use model by Bill Murray, SCDsource 2008.
11. Post-Silicon Debug Using Formal Verification Waypoints by C. Richard Ho, Michael Theobald, Brannon Batson, J.P. Grossman, Stanley C. Wang, Joseph Gagliardo, Martin M. Deneroff, Ron O. Dror, David E. Shaw. Proceedings of the 43rd Design Automation Conference, 2006.
12. Can We Really Do Without the Support of Formal Methods in the Verification of Large Designs? Umberto Rossi. 42nd Design Automation Conference, 2005.
13. Assertion-Based Design. Harry Foster, Adam Krolnik, David Lacey. Springer Verlag, 2004.
14. Unified Coverage Database Application Programming Interface (API) Design Specification Document, Draft 29, April 2, 2009. Accellera Unified Coverage Interoperability Standard (UCIS) committee.
15. Formal Techniques Speed Up Interconnect Verification of SystemC Virtual Platform Models. Wolfgang Ecker, Volkan Esen, Robert Schwencker, Thomas Steininger, Michael Velten. DVCon 2008. Reported in "Formal Verification Expands Its Use Model by Bill Murray, SCDsource 2008.

Further Reading

1. Formal Property Checking - What the Users Say, by Richard Goering, SCDsource 2008
2. Increasing Confidence of Complex Hardware in Safety-Critical Avionics Using Formal Methods. Kristoffer Karlsson and Håkan Forsberg. Military and Aerospace Programmable Logic Devices (MAPLD) Conference 2006.