Automated connectivity checking with formal verification

Tom Anderson

Formal verification traditionally has been regarded as an advanced technique for experts to thoroughly verify individual blocks of logic, or perhaps small clusters of blocks. The appeal of formal techniques is the exhaustive analysis of all possible behavior for the design being verified. This stands in sharp contrast to simulation, which exercises only a tiny fraction of possible behavior by running specific tests. If no test triggers a design bug, the bug will not be found. If the bug is triggered but no change in results is observed, the bug will not be found. Given a sufficiently robust set of properties to describe intended behavior, formal tools can not only find all bugs but also prove that there are no more bugs to be found.

Today, many more users can take advantage of the power of formal verification, and most of them are not experts. There are several reasons why formal adoption has grown so much. The broad deployment of standardized formats, most notably the SystemVerilog Assertions (SVA) subset, has reduced the level of expertise needed to write formal properties. Model-based mutation coverage can identify those parts of the design not covered by assertions, providing valuable guidance to users. Formal tools now have more automation and simulation-like debug features, making them easier to use. Regular breakthroughs in the power and performance of formal algorithms enable use on large blocks and clusters unimaginable just a few years ago.

However, the primary reason for the wider use of formal verification is that the majority of users are running applications ("apps") targeted for specific verification challenges. Apps typically generate most, or all, of the properties needed for formal analysis, with algorithms and tool features tuned for the target application. The result is a "pushbutton" solution requiring minimal training even for users with no formal experience. Further, apps are so efficient that many are run at the full-chip level even for very large system-on-chip (SoC) designs. The goal remains finding all bugs and proving that all bugs have been found, but only those bugs related to the specific challenge being addressed.

SoC connectivity challenges

Connectivity checking is one of the most widely used application for formal technology. The purpose of this verification task is deceptively simple to state: Ensure the proper interconnections among design blocks and I/O cells. This sounds easy enough, but in fact it is a significant challenge. A modern SoC contains complex subsystems built with thousands of instances of highly configurable modules and IP blocks. Programmable elements provide flexibility and adaptability, while multiplexed I/O pads allow user control of which protocols run on which pins. There may be hundreds of thousands of connecting paths, every one of them important for proper functional operation of the chip.

Signals that are supposed to be connected may go through multiple blocks and multiple levels of hierarchy, as shown in figure 1. Inverters may exist along the paths, so it is critical to track polarity. Paths may also include state elements such as registers and flip-flops, resulting in multi-cycle delays between starting and ending points. Some global signals such as clocks, resets, and scan enables are routed to thousands or millions of state elements, and the correctness of these connections should also be verified. All these reasons, plus the sheer number of connections to be checked, render connectivity verification by inspection completely impractical.

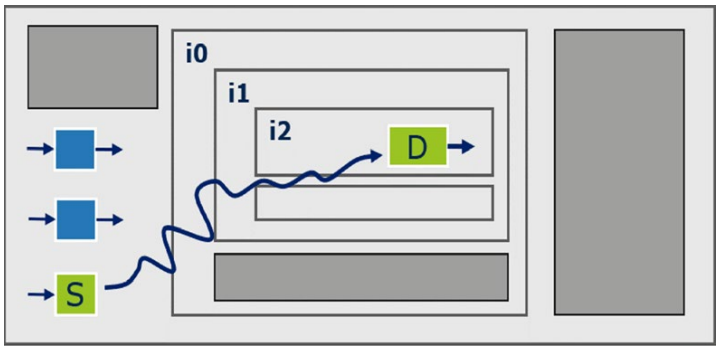

Fig. 1: Connectivity source and destination may be separated by levels of hierarchy.

Simulation or emulation of connectivity is more practical, but inherently incomplete. A test suite to cover every path required would be tedious to write, difficult to maintain as the design evolves, and time-consuming to run. Debug when errors are detected is not trivial; bug symptoms must be traced back to find the incorrect or missing connection. Of course, neither simulation nor emulation can provide any sort of proof of correctness, no matter how good the test suite may be. This is precisely why connectivity checking has become such a widely used application for formal technology. A formal tool can potentially find all the connection errors and prove that connectivity is complete after all bugs have been fixed.

Traditional formal connectivity checking

As previously noted, formal verification requires properties against which the design is checked. It is not hard to imagine writing a series of properties using SVA to specify the signals that must be connected, and this might be a tractable approach for small designs. Writing thousands of such properties would be daunting, and this is a key area where a connectivity checking formal app can help. The structure of connectivity properties is quite regular, so they can be generated automatically given a specification of intended connectivity. This is most often provided to the formal tool in the form of a traditional spreadsheet, as shown in figure 2.

Clearly, filling out this spreadsheet is much easier for users than writing assertions. The fields specify the source and destination for each connection path, the number of cycles of delay along the path, the condition under which the path should be enabled, and the relevant clock. The enabling condition is

Tom Anderson is technical marketing consultant at OneSpin Solutions - www.onespin.com

```
# Source          # Destination     # Delay # Condition          # clock
top.a.b.c,        top.x.y.z
top.i1_m_s_bit.x, top.i1_m_d_bit.y, 1,      top.debug_en==1'b0, top.clk
top.i2_m_s_bit.x, top.i2_m_d_bit.y, 2:3,    top.debug_en==1'b0, top.clk
top.i1_m_s_bot.x, top.i1_m_d_bot.y, 5,      top.debug_en==1'b0, top.clk
top.i2_m_s_bot.x, top.i2_m_d_bot.y, 2,      top.debug_en==1'b0, top.clk
```

Fig. 2: Connectivity intent can be specified in a spreadsheet.

especially critical for paths containing multiplexors, such as I/O pads supporting multiple possible connections under different conditions. Given the information in the spreadsheet, a formal tool can generate all properties required with no manual specification. A combination of structural analysis and formal proof engines finds all bugs in the design (or errors in the spreadsheet) and then proves full conformance to the specification.

Like several other formal apps, connectivity checking is routinely run on full-chip designs. This is necessary since the full range of connections to be verified is visible only at the top level. Formal tools have capacity limitations, but connectivity checking is possible on large chips because only a small portion of the design is relevant to the problem at hand. Unrelated logic is trimmed away while building the formal model to speed analysis. However, today's very large heterogeneous computing platforms and similar SoCs stress the capacity of traditional formal tools. Further, filling out a spreadsheet with hundreds of thousands of entries rather than thousands is not realistic. Clearly, traditional formal connectivity checking must evolve.

The connectivity XL approach

A new methodology for connectivity verification, dubbed Connectivity XL by OneSpin Solutions, addresses the challenges of massive SoC designs. One of the key innovations is the elevation of connectivity intent specification to an abstract level. As shown in Figure 3, a spreadsheet remains the vehicle, but wildcards make the specification much more concise.

```
# Source                  # Destination
i:top.a.b, sig_c,         m:m_x,      sig_y
m:m_a,     sig_b,         i:top.x,    sig_y
m:m_s_b?p, sig_a,         m:m_d_b?p,  sig_x
m:m_s_b?t, sig_b,         m:m_d_b?t,  sig_y
m:m_s_b?o, sig_c,         m:m_d_b?o,  sig_z
m:m_s_*t,  sig_d,         m:m_d_*t,   sig_x
```

Fig. 3: Abstract connectivity specification is concise.

It is common for blocks to be instantiated multiple times with regular naming, so wildcards can compress the required number of spreadsheet lines significantly. This can reduce the time to specify intended connectivity from months to days.

A formal tool can read this abstract specification, compile it together with the design, and expand the wildcards to produce a traditional connectivity spreadsheet with a single connection per line. However, this specification may have hundreds of thousands of lines, so a traditional connectivity checking tool would likely have capacity issues. Ongoing improvements in the underlying formal algorithms of Connectivity XL support ever-longer connectivity specifications for ever-larger SoC designs. Machine learning based on many years of formal experience is used to select the best proof engine for the job.

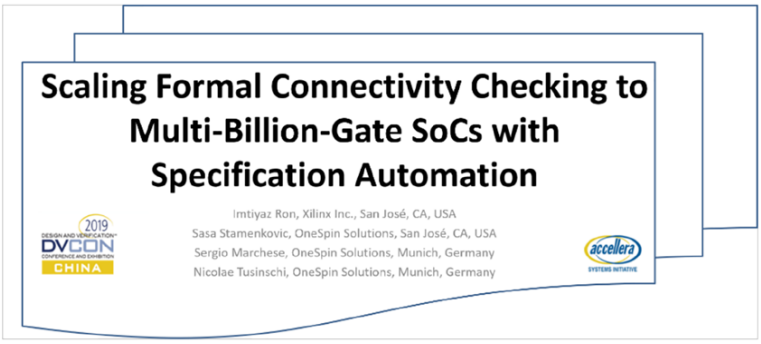

Fig. 4: A connectivity case study was presented at DVCon China 2019.

Automatic abstractions reduce the formal model to the minimal require logic, speeding up runtimes and reducing memory usage. Another innovation of Conne ctivity XL is unifying structural and formal analysis for maximum efficacy. As part of generating detailed specifications, this analysis automatically detects delays and inverters in the connection paths and infers multiplexing conditions. In summary, Connectivity XL provides a more automated flow than traditional approaches, handles larger designs, and produces complete proofs even for the most complex chips.

Real-world verification results

At the recent Design and Verification Conference (DVCon) in China, Xilinx and OneSpin presented a case study of the OneSpin Connectivity XL App applied to a multi-billion-gate SoC. Using 7 nm technology, this chip contained 60 million instances of 35 thousand modules, 90 million flip-flops, and 80 thousand finite state machines. As one of the largest designs in the world, it stressed many tools in the design and verification flows. This was certainly true for connectivity checking since there were in excess of one million connections to specify, maintain across design iterations, and verify.

The verification team tried several traditional connectivity apps, including OneSpin's, and all failed to scale to this large chip. The effort to specify and maintain more than a million connections was unacceptable. Formal tool runtimes were excessive, and too often produced inconclusive proof results. With a tight design schedule, quality could not be compromised, and exhaustive verification was deemed critical. Connectivity XL proved to be up to the task. The abstract specification format reduced spreadsheet size by a factor of more than one hundred while making it easier to maintain the connection list.

Connectivity XL found several corner-case bugs that would have been very hard to detect using any other tool or method. The errors included incorrect block integration, multiple drivers enabled on paths, and re-convergent paths. The debug information provided enabled easy root-causing, even on paths with more then two thousand signals between source and destination. Once these issues were resolved, all one-million-plus connections were proven within a matter of days using multiple jobs running in parallel. There were no inconclusive results for any connections.

Conclusion

Ever-increasing chip size and complexity is making formal apps even more valuable, especially connectivity checking. There is no chance that simulation, emulation, or manual techniques will suffice. Even traditional formal tools do not scale. The Connectivity XL approach is the next generation of connectivity checking solution, with both greater capacity and improved automation. It has been validated on a real-world multi-billion-gate SoC design with more than a million connections. Designs will continue to grow but this new category of formal tools is positioned to provide a viable solution for years to come.

Connecting with e-textiles: Find the box!

By James Hayward

Commercial efforts around electronic textiles have been prominent for at least 25 years, starting with early patents and then early products throughout the 1990s. Electronic components including batteries, transistors/microprocessors, antennae for communication and so on, have all been demonstrated in a textile format, and examples of these are included in IDTechEx' latest research report "E-Textiles 2019-2029: Technologies, Markets and Players".

The majority of these demonstrations are a one-off proof of concept, and certainly not commercially mature enough for wider deployment. Therefore, as nearly all e-textile products will need these components, commercial options today typically include traditional rechargeable batteries and housed PCBs containing the other essential electronic components. The result is that these components need to be housed somewhere in the e-textile product, and hence it is typically possible to "find the box" which contains these components.

This electronic box can potentially be a good solution to the challenge of washing. Typically, e-textile products can be sold with multiple versions of the garment element and a single box to fit all of them. Then the box can be removed for washing and replaced onto copies of the garment element, just as would be the case for a smartwatch or chest strap. This requires a replaceable, reliable, durable and fool-proof connector option, for which traditional snap fasteners or magnetic versions are typically preferred. As our report shows, the industry has