Seven Steps for IoT Verification

Many IoT devices are complex system-on-chip (SoC) designs with embedded software. The development isn't trivial, and the verification is critical for successful deployment in end products.

TOM ANDERSON *

There are few topics more discussed in the electronics press than the Internet of Things (IoT), the vision that almost everything in our world will be interconnected. IoT enthusiasts can walk through a home and point to dozens of potential examples, from smart home-security and climate-control systems to Internet-enabled refrigerators and pantries that know how to reorder items based on electronic tags in food packaging. A walk through an office or a factory floor would yield similar examples of possible IoT connectivity. It is unclear how much of this broad vision will come true, and which IoT devices will turn out to be genuinely useful rather than fads. But there is no arguing the general trend that more and more of our world will be interconnected, tying our time at home, at work, and in transit ever closer together. Chip manufacturers have scrambled to produce devices suitable for IoT applications, and there seems no limit on how many types of end products electronics systems manufacturers can invent.

Security first:
In the vision of the Internet of Things almost everything will be interconnected. The verification of the often complex systems is key for security. Source: Clipdealer

Many IoT devices are complex system-on-chip designs

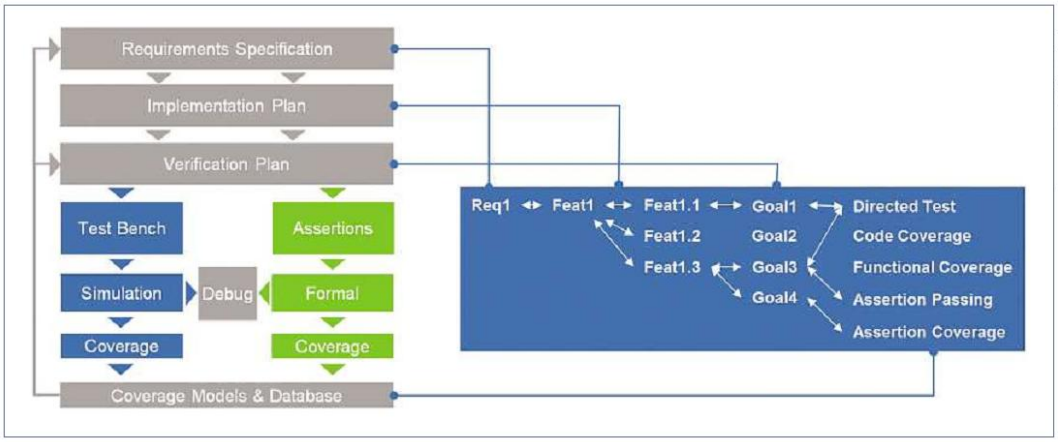
Some IoT developers believe that their jobs are easier than those of their colleagues who design and verify chips for smart phones, big data servers, and artificial intelligence applications. This is not necessarily true. Many IoT devices are complex system-on-chip (SoC) designs with embedded software. The latest chips from ARM and Intel's Atom family, both considered suitable for IoT, contain multiple processor cores and complex specialty engines. This is especially true for chips on the high end of IoT, such as those for autonomous vehicles. Some developers also believe that the bar for verification of IoT devices is lower than for other types of chips. Their assumption is that troublesome hardware is simply thrown away and replaced with a new version. But many IoT devices are in hard-to-access locations. It is not at all easy to repair or replace a rooftop security camera, environmental sensors scattered throughout a building, or components of an active factory production line. It is far better to find and fix all bugs prior to shipping and installing such end products.

Further, the assumption that lurking bugs or faults in the field are less critical for IoT chips is flawed. Of course, chips for vehicular applications have very high verification requirements indeed. But even for simpler devices consumer expectations are high. No one wants to be shut out of the office by a malfunctioning door lock or vulnerable at home when all security cameras and sensors are down. Even cell phones must be more reliable; in the absence of land-line telephones they may be the only way to summon help in an emergency.

Verification of IoT chips is critical for successful deployment

For all these reasons, verification of IoT chips is now critical for successful deployment in end products. There are seven key steps to this process that should be followed by every IoT verification engineer: module-level static analysis, module-level formal analysis, chip-level simulation, chip-level

*** Tom Anderson**
... is technical marketing consultant at OneSpin Solutions.



Sources: OneSpin Solutions

Figure 1: *Verification planning assigns different tools for different parts of the design and then combines results.*

formal analysis, equivalence checking, safety verification, and hardware/software co-verification. These same seven steps are required regardless of whether the IoT chip uses full custom, ASIC, or FPGA technology. Modern FPGAs are also SoCs, requiring the same verification rigor as ASICs. Static analysis is the starting point for fresh RTL code, especially at the module level, although much of this analysis can also be run on large blocks and even full chips. A comprehensive "linting" tool that looks for common coding errors is essential. Whole classes of simple bugs can be detected and eliminated. Examples include synthesis-simulation mismatches, bus contention, x-value propagation, incorrect signal connectivity, and register implementation or bus protocols not matching the specifications.

The full power of formal methods unleashed

The more advanced of these static checks entail the use of formal engines to achieve the deep levels of analysis required for exhaustive results. However, the formal applications ("apps") that perform these checks do not require manual specification of assertions or any knowledge of formal verification. Any assertions needed are generated within the apps themselves. The full power of formal methods is unleashed when the development team adopts assertion-based verification (ABV), including specification of assertions to document design intent. Many RTL designers are quite comfortable writing assertions, and in fact find them much easier than developing an object-oriented simulation test bench. Designers typically do some formal runs but rely on the verification team to help with constraints, abstractions, and other aspects requiring deeper formal knowledge. Some types of modules are verified using formal ABV only, with no simulation until the chip level. As modules are combined into larger blocks, formal analysis can be selectively re-run to verify that the interaction among the modules has not introduced design errors.

At the full-chip level, simulation (step three in IoT verification) is still common. A modern, constrained-random, object-oriented test bench can exercise the design far more thoroughly than hand-written tests. Almost all chip-level simulation today uses SystemVerilog and adheres to the Universal Verification Methodology (UVM) standard. Commercial verification IP (VIP) vendors provide a wide range of building blocks to help create the test bench, especially for on-chip-bus and external interface standards. Formal analysis also has a major role to play at the chip level. Many types of verification performed by formal apps, although also run on smaller blocks, must be run on the full chip since only at the top level is the complete design visible. These apps check for correct signal connectivity, including connections to the chip pins, proper synchronization across clock-domain crossings, and legal combinations of power domains on, off, or running at reduced voltage or frequency. No amount of simulation can match the exhaustive nature of formal verification for these and other checks.

Clearly, the combination of formal analysis and simulation is a requirement for thorough functional verification of an IoT chip. Therefore, deciding which approach to use for which part of the design should occur early in the planning process. As shown in Figure 1, design requirements from the product specification are mapped to individual features, each of which is verified using the chosen approaches and tools. The results from all the tools are combined, essential for assessing verification progress and deciding what to do next.

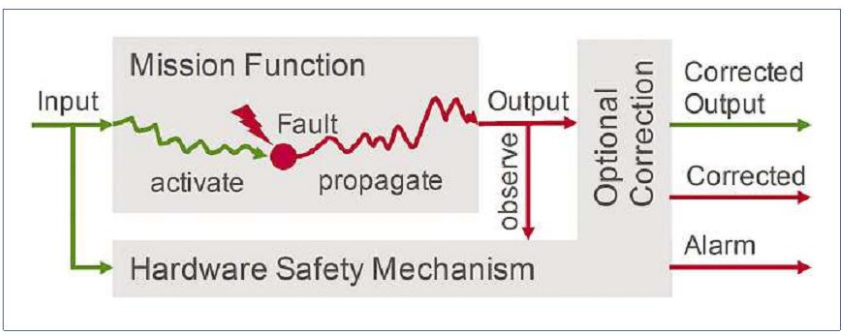


Figure 2: *Errors in safety-critical designs must be detected and then corrected if possible.*

Different tools for different parts of the design

At this point, the first four steps of IoT verification are complete. Ideally, all RTL design bugs have been found and fixed. But even if this is true, the work of the verification team is not done. Many chips must conform to functional safety requirements as specified in such standards as IEC 61508 and ISO 26262. Even if there are no design bugs remai-

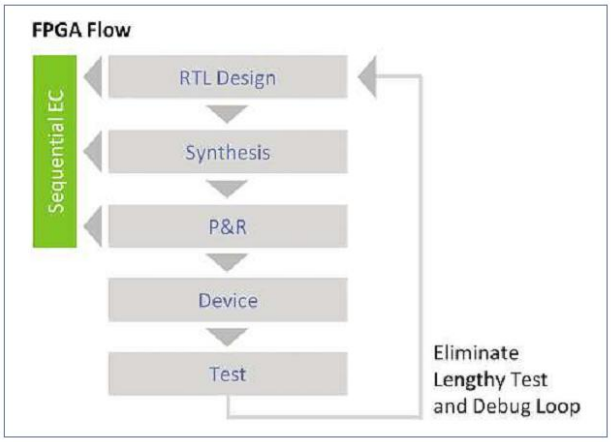


Figure 3: *Formal equivalence checking is essential to ensure that an FPGA design remains.*

ning, a chip must continue to work in the field even after a random error such as an alpha particle flipping a bit in a memory.

To guard against this possibility, an IoT chip may contain a hardware safety mechanism that detects when a random fault has altered the output of the design. At a minimum, an alarm must be raised so that corrective action may be taken. For example, an autonomous vehicle might pull safely to the side of the road and reboot the system in the hopes of clearing the problem. It is even better if the erroneous behaviour can be corrected, with the error logged for diagnostics and analysis later. Figure 2 shows this process. Formal analysis plays a key role in safety verification. Given an appropriate fault model, faults can be inserted and analyzed to see whether they propagate to an output. Further analysis determines whether these propagated faults are detected by the hardware safety mechanism, and the percentage of the design protected against random faults can be calculated. Some safety standards require a high percentage of protection, so a robust and automated analysis process is essential.

Simulation of the RTL design and the final netlist is to slow

Step six of IoT verification considers a different problem: errors introduced into the design by test insertion tools, logic synthesis, or the place-and-route process. Simulation of the RTL design and the final netlist side by side is slow, expensive, and may not find errors. Only the exhaustive nature of formal equivalence checking can be certain that the tools have not changed the design in some unintended way. Equivalence checks can be performed on each step of the design flow or on the initial RTL and the placed-and-routed netlist. As noted earlier, modern FPGA designs require the same level of rigor as ASICs. When it comes to equivalence checking, the requirements are actually higher for FPGAs. To extract maximum performance and capacity from the underlying technology, FPGA design flows perform optimizations that change the inner structure of the design, for example by reordering state machines or moving logic from one side or a register to the other. Only formal sequential equivalence checking can really ensure that the overall functionality is unchanged, as you one see in Figure 3.

Verification shortcuts are not appropriate for IoT designs

The final step in IoT verification is running the hardware and software together in an environment modeling the real world as much as possible. This is usually accomplished by mapping the chip design into an in-circuit emulation (ICE) or FPGA-based prototyping system, running product software on the embedded processors, and plugging the whole system into the socket that will eventually hold the fabricated chip. This hardware-software co-verification runs much faster than simulation and exercises the hardware-software boundary.

Executing these seven steps may seem like quite a tall order for verifying a chip that many see as simple and disposable. As noted, many IoT devices are complex, hard to replace, and used in applications with high reliability and safety requirements. Verification shortcuts are not appropriate. No one wants a "smart" freezer full of rotting food or an astronomical heating bill because the climate-control system failed. IoT development teams must follow all seven steps to minimize the chances of embarrassing or even dangerous results. // **ME**

OneSpin Solutions