

A Methodology for Writing and Exploiting Formal Specifications

Kanna Shimizu
Stanford University

Many of today's digital circuit designs depend on the tight integration of multiple design components. These components are often designed by different engineers who may have divergent interpretations of the interface protocol that "glue" the components together. Consequently, designs may be incompatible and behave incorrectly when combined. Thus, functional interface specifications are pivotal for successful module integration and should be solid and precise accordingly.

However, specifications widely in use today are still written informally, forfeiting an opportunity for automated analysis and logical clarity. In many cases, specifications such as those of standard bus protocols are buggy, ambiguous, and contradictory: all problems that can be resolved by formal specification development.

The advantages of formal specifications seem clear but formal specifications are nonetheless avoided due to a perceived cost-value problem; they are often considered too costly for the benefits they promise. Specifically, they are criticized for their lengthy development time and the investment needed for formal verification training. For many, the value of a correct specification does not justify these costs, and precious resources are allocated for more pressing design needs. Thus, to counter these disincentives, a formal specification methodology that attacks this cost-value problem from two angles was developed.

- **Cost Side** Develop a regimented specification style that leads to a correct specification with less effort than free-form, ad hoc methods.

- **Value Side** Increase the value of a formal specification beyond its role as a documentation. Clearly, if the specification shortens implementation design time from various angles, the investment in a formal specification would be more acceptable.

Cost Solution : Structured Specification Writing As examples of interface protocols, signal-level descriptions for standard bus protocols were used since they are both important and challenging to specify. Formal specifications for a core subset of the PCI (Peripheral Component Interconnect) protocol and the Intel[®] Itanium[™] Processor bus protocol were developed using our specification style. The style is based on writing many compact properties to collectively describe protocol behavior and avoiding large state machines and complex constructs (such as time-unbounded or non-deterministic constructs). The syntactic structuring used in the style is language-independent and can be applied to specifications written in many languages from SMV to Verilog. This is because methodology, as opposed to tool or language development, is the key to achieving the cost-value goal.

At the core of the methodology is the idea of constraining the way a specification can be written. By restricting the use of the specification language and by enforcing a grammatical structure to the description, a specification attains many desirable properties (for both writing and exploiting the specification). Although this lack of flexibility may prevent certain characteristics to be described, the restrictions were not prohibitive for the two protocols mentioned above. Even pipelining was handily described. Because of this structuring, the specification is executable, easy to write for engineers with little formal verification training, and is provably implementable among other attributes. Furthermore, debugging methods specific to the style can find contradictions and other bugs in the protocol even before implementations are built. With both example protocols, previously undiscovered issues such as deadlock and dropped data phases were found by these methods.

Value Solution : Unified Framework Approach Before a verification engineer can start simulating RTL designs, he must write three *verification aids*: input testbenches to stimulate the design, properties to verify the behavior, and a functional coverage metric to quantify simulation progress. Note that all three are fundamentally based on the interface protocol. However, in practice, they are developed separately resulting in overlapping efforts. To counter this inefficiency, we propose using a complete interface specification to automatically derive all three. Compilers were written to create a real-time input generator, an output checker, and a coverage metric, all from the specification. Others have used similar techniques in isolation, but this *unified framework approach*, centered around the interface specification, is a unique contribution. Furthermore, compared to prevalent input generation schemes, the specification structuring described above allows for *dramatically* smaller memory usage. To test these ideas, a PCI specification was used to simulate and verify a large-scale I/O design from the Stanford FLASH project. In the process, previously unreported bugs in the design were found.