AFTERWORD

Two questions that often arise from customers are:

1. For verification tasks, should we transition away from Verilog or VHDL and into SystemVerilog? Our concerns are the complexity of SystemVerilog as it offers a spectrum of features, and that can come at the expense of an expensive learning curve.
2. How do we use all of the rich features of SystemVerilog for verification? Individually, we can figure the use of the advanced constructs, but how do we create a cohesive, simple structure around this extensive language to create a reusable, maintainable testbenches that is common in style among the many design groups?

After hours of discussions, proposals, cost analysis, meetings, and salesmanship, verification engineers do eventually convince management on the adoption of SystemVerilog because it is indeed a very powerful language suitable for verification. But adopting SystemVerilog is half the battle in supporting the verification environment. The other battle is defining that verification framework.

…And therein lies the problem! Many verification engineers have hardware versus software background. Without access to a well defined and verified framework like VMM, individual organizations will develop their ingrown techniques in the use of SystemVerilog for testbenches. These techniques will most likely be an adaptation or a translation of existing individualized, non object-oriented programming techniques inherited from previous projects. After all, that's the easy path, and they worked in the past! The use of a new language a la "old" Verilog or VHDL style is not sufficient to verify today's complex designs. All the benefits that SystemVerilog provide are not fully taken advantage of. Some organizations with software expertise may begin to adopt the advanced features of SystemVerilog by constructing a framework in an ad hoc manner. The end result of these various adaptations is that they tend to be chaotic at the whim of every organization, thus bringing in the disorder that management originally wanted to avoid.

This is where VMM really shines. VMM provides an easy to use framework above SystemVerilog, and provides advanced object-oriented verification features with consistency in design style, flexibility, and reuse. The VMM framework puts the verification effort where it belongs, in the definition of the verification rather than the detailed coding of the libraries to support the verification. VMM supports the reusable and extendable techniques with object-oriented patterns such as factories and callbacks. VMM also supports the definition of the reusable environment and the stepping of the various simulation tasks such as *reset, configuration, start, run, wait_for_end, cleanup* and *report*. The reporting features supported by VMM enables design specific reporting, but yet maintains a consistent style in creating that reporting environment.

Our experience with VMM is very positive because it requires a minimal knowledge of object-oriented programming to put the verification effort where it belongs, in the problem at hand. It also brings to the design of the verification environment a common feel and look across all designs with a framework that has endured years of maturity.[1]

[1] VMM is the SystemVerilog adaptation of RVM (Reuse Verification Manual) that used the proprietary VERA language.